# Defining Benchmarks for Continual Few-Shot Learning

**Antreas Antoniou** [1] **Massimiliano Patacchiola** [1] **Mateusz Ochal** [1] **Amos Storkey** [1]

## Abstract

Both few-shot and continual learning have seen substantial progress in the last years due to the introduction of proper benchmarks. That being said, the field has still to frame a suite of benchmarks for the highly desirable setting of *continual few-shot learning*, where the learner is presented a number of few-shot tasks, one after the other, and then asked to perform well on a validation set stemming from all previously seen tasks. Continual few-shot learning has a small computational footprint and is thus an excellent setting for efficient investigation and experimentation. In this paper we first define a theoretical framework for continual few-shot learning, taking into account recent literature, then we propose a range of flexible benchmarks that unify the evaluation criteria and allows exploring the problem from multiple perspectives. As part of the benchmark, we introduce a compact variant of ImageNet, called *SlimageNet64*, which retains all original 1000 classes but only contains 200 instances of each one (a total of 200K data-points) downscaled to $64 \times 64$ pixels. We provide baselines for the proposed benchmarks using a number of popular few-shot learning algorithms, as a result, exposing previously unknown strengths and weaknesses of those algorithms in continual and data-limited settings.

## 1. Introduction

Two capabilities vital for an intelligent agent with finite memory are *few-shot learning*, the ability to learn from a handful of data-points, and *continual learning*, the ability to sequentially learn new tasks without forgetting previous ones. There is no question that modern machine learning methods struggle in combining these two capabilities, while humans and animals possess them innately.

One of the main reasons behind the scarce consideration of the liaison between the two is that these problems have been often treated separately and handled by two distinct communities. Historically the research on continual learning has focused on the problem of avoiding the loss of previous knowledge when new tasks are presented to the learner, known as *catastrophic forgetting* (McCloskey & Cohen, 1989), without paying much attention to the low-data regime. On the other hand, the research on few-shot learning has mainly focused on achieving good generalization over new tasks, without caring about possible future knowledge gain or loss. Scarce attention has been given to few-shot learning in the more practical continual learning scenario.

Taken individually these two areas have recently seen dramatic improvements mainly due to the introduction of proper benchmark tasks and datasets used to systematically compare different methods (Chen et al., 2019; Lesort et al., 2019a; Parisi et al., 2019). For the set-to-set few-

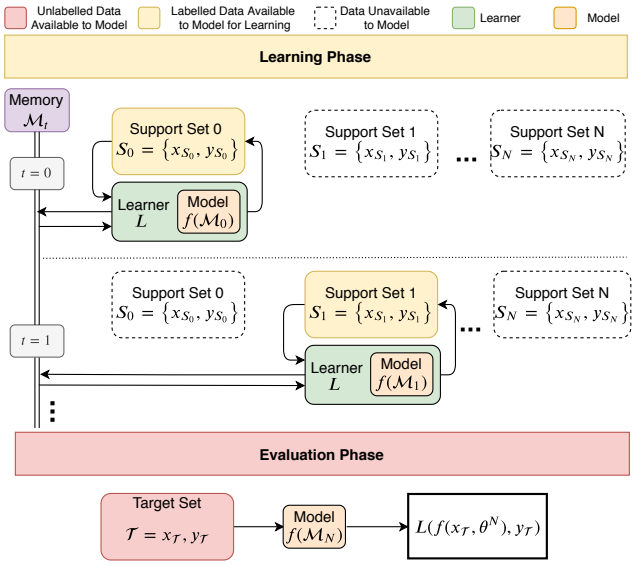


Figure 1: High level overview of the proposed benchmark. Top block: from the left, a learner acquires task-specific information from each set, one-by-one, without being allowed to view previous or following sets (memory constraint). The learner can store that knowledge in a shared memory bank. The stored knowledge can be used by a given classification model. On the rightmost side, future tasks are inaccessible to the learner. Central block: the same process is repeated on the second support set. Note that the first support set is now inaccessible. Bottom block: once the learner has viewed all support sets, it is given an evaluation set (target set) containing new examples of classes contained in the support-sets, and tasked with producing predictions for those samples. The evaluation procedure has access to the target set labels and can then establish a generalization measure for the model.

[1]School of Informatics, University of Edinburgh. Correspondence to: Antreas Antoniou <a.antoniou@sms.ed.ac.uk>.

shot learning setting (Vinyals et al., 2016) such benchmarks include Omniglot (Lake et al., 2015), CUB-200 (Welinder et al., 2010), Mini-ImageNet (Vinyals et al., 2016) and Tiered-ImageNet (Ren et al., 2018), whereas for the single-incremental-task continual learning setting (Maltoni & Lomonaco, 2019) and the multi-task continual setting (Zenke et al., 2017; Lopez-Paz & Ranzato, 2017) the benchmarks include permuted/rotated-MNIST (Zenke et al., 2017; Goodfellow et al., 2013), CIFAR10/100 (Krizhevsky et al., 2009), and CORe50 (Lomonaco & Maltoni, 2017). However, none of those benchmarks are particularly well suited for the constrained task of learning on low-data streams.

Few-shot learning focuses on learning from a small (single) batch of labeled data points. However, it overlooks the possibility of sequential data streams that is inherent in many robotics and embedded systems, as well as standard deep learning training methods, such as minibatch-SGD, where what we have is effectively a sequence of small batches from which a learner must teach an underlying model. On the other hand, continual learning is a broad field encompassing many types of tasks, datasets and algorithms. Continual learning has been applied in the context of general classification (Parisi et al., 2019), video object recognition (Lomonaco & Maltoni, 2017), and others (Lesort et al., 2019c). Most of the investigations are done on continual tasks of very long lengths, using relatively large batches. Moreover, each sub-field has their own combinations of variables (e.g. size and length of sequences) and constraints (e.g. memory, input type) that define groups of continual learning tasks. We argue that our proposed setting helps to formalize and constrain an emerging group of tasks within a low-data setting.

In this paper we propose a setting that bridges the gap between these settings, therefore allowing a spectrum starting from strict few-shot learning going in the middle to short-term continual few-shot learning and on the other end arriving at long-term continual learning. We propose doing this by injecting the sequential component of continual learning into the framework of few-shot learning, calling this new setting *continual few-shot learning*. While we formally define the problem in Section 3, a high-level diagram is shown in Figure 1.

In addition to bridging the gap, we argue that the proposed setting can be useful to the research community for four additional reasons. **1.** As a framework for studying and improving the sample efficiency of mini-batch stochastic methods. Mini-batch training is quite inefficient computationally, because it requires multiple learning iterations over a dataset to learn a good model. **2.** As a minimal and efficient framework for studying and rectifying catastrophic forgetting. Improvements can come in two flavors, either via meta-learning models which can provide insight into better learning dynamics, or by designing general methods to rectify the problem. **3.** As a framework for studying continuous adaptations of neural networks under memory constraints (e.g. robotics, embedded devices) **4.** Due to its continual length and small batch size, CFSL is ideal for investigating and training meta-learning systems that are capable of continual learning. We have made sure that all our settings fit on a single GPU with 11 GBs of memory.

Our main contributions can be summarized as follows:

1. We formalize a highly general and flexible continual few-shot learning setting, taking into account recent considerations and concerns expressed in the literature.
2. In order to foster a more focused and organized effort in investigating continual few-shot learning, we propose a new benchmark and a compact dataset (SlimageNet64), releasing them under an open source license.[1]
3. We compare recent state-of-the-art methods on our proposed benchmark, showing how continual few-shot learning is effective in highlighting the strengths and weaknesses of those methods.

## 2. Related Work

### 2.1. Few-Shot Learning

Progress in few-shot learning (FSL) was greatly accelerated after the introduction of the set-to-set few-shot learning setting (Vinyals et al., 2016). This setting, for the first time, formalized few-shot learning as a well defined problem paving the way to the use of end-to-end differentiable algorithms that could be trained, tested, and compared. What followed was an explosion of progress in the field. Among the first algorithms to be proposed there were meta-learned solutions, which here we group into three categories:

**Embedding-learning and Metric-learning:** Those methods include the Neural Statistician (Edwards & Storkey, 2017), Matching Networks (Vinyals et al., 2016) and Protoypical Networks (Snell et al., 2017). They are based on the idea of parameterizing embeddings via neural networks and then use distance metrics to match target points to support points in latent space. The whole process is fully differentiable and it is trained such that the model can generalize to a wide range of tasks.

**Optimization-based or Gradient-based Meta-Learning:** Those methods have been introduced in the form of MetaLearner LSTM (Ravi & Larochelle, 2016), MAML (Finn et al., 2017), Meta-SGD (Li et al., 2017) and MAML++ (Antoniou et al., 2019). The model itself is a model for learning,

[1] Available from https://zenodo.org/record/3672132

explicitly trained to achieve a particular set of tasks. More specifically, in such models there is an inner-loop optimization process that is partially or fully parameterized with fully differentiable modules. This inner-loop process is optimized such that if a model uses it to learn from a support set, then it will generalize to a target set. The process that learns the learner is the outer-loop optimization process. This mechanism of learning to learn, is often called *meta-learning* (Schmidhuber, 1987). Recent methods such as LEO (Rusu et al., 2019) and SCA (Antoniou & Storkey, 2019) have combined both categories to create very strong state-of-the-art systems.

**Hallucination-based:** Those algorithms can utilize one or both the aforementioned methods in combination with a generative process, to produce additional samples as a complement to the support set. An example of this approach has been recently presented by Antoniou et al. (2017).

**Other solutions:** There have been a number of methods that do not clearly fall in one of the previous categories. One example are Bayesian approaches, like those based on amortized networks (Gordon et al., 2019), hierarchical models (Grant et al., 2018), or Gaussian Processes (Patacchiola et al., 2019). Another example are Relational Networks (Santoro et al., 2017), originally created to deal with relational reasoning; they have been adapted to the few-shot learning setting with good performance (Santurkar et al., 2018). In addition, simpler approaches such as pretraining of a neural network on all classes and fine tuning on a given support set, have also shown to perform fairly well (Chen et al., 2019). Similarly, a method based on nearest neighbor classifier has recently showed to achieve state-of-the-art performances (Wang et al., 2019).

### 2.2. Continual Learning

The problem of continual learning (CL), also called lifelong learning, has been considered since the beginnings of artificial intelligence and it remains an open challenge in robotics (Lesort et al., 2019c) and machine learning (Parisi et al., 2019). In standard supervised learning, algorithms can access any data point as many times as necessary during the training phase. In contrast, in CL data arrives sequentially and can only be provided once during the training process. Following the taxonomy of Maltoni & Lomonaco (2019) we group the continual learning methods into three classes: architectural, rehearsal, and regularization methods.

**Architectural methods:** Architectural strategies add, clone, or save parts of trained weights (Lesort et al., 2019a). For example, progressive neural networks (Rusu et al., 2016) create a new neural network for each new task and connect it to previously generated networks, thus leveraging previously learned knowledge while solving catastrophic forgetting. Another architectural strategy includes weight freezing (Mallya et al., 2018; Mallya & Lazebnik, 2018) where some weights are frozen dynamically to retain knowledge of old tasks, while leaving others to freely adapt to new tasks later on.

**Rehearsal methods:** Rehearsal strategy methods selectively choose which data points to store within a bounded amount of resources. One such algorithm stores top-$N$ most representative samples of a class while maintaining a fixed upper bound on the required memory (Rebuffi et al., 2017). More recently, generative models such as GANs and VAEs (Lesort et al., 2018; 2019b) have been proposed to represent previously seen data as weights of a neural network.

**Regularization methods:** Unlike other approaches, regularization methods focus on adding constraints on parameter updates of neural networks to directly minimize catastrophic forgetting. For example, Elastic Weight Consolidation (EWC, Kirkpatrick et al. 2017; Mitchell et al. 2018) slows down the learning rate of those weights that are responsible for previously learned tasks. Other regularization techniques have been recently presented which follow a similar approach (Zenke et al., 2017; Lee, 2017; He & Jaeger, 2018).

All of the outlined approaches offer various advantages and disadvantages under resource constraints. Architectural approaches can be constrained on the amount of available RAM, whereas, rehearsal strategies can become quickly bounded by the amount of available storage. Regularization approaches can be free from resource constraints but incur in severe issues in the way they adapt model parameters. Note that the outlined strategies are not mutually exclusive and can be combined (Rebuffi et al., 2017; Maltoni & Lomonaco, 2019; Kemker et al., 2018).

*Online learning* is a special case of CL where new data becomes available a single data point at a time. *Active learning* can also appear in continual learning settings but it is a special type of semi-supervised machine learning, that aims to strategically select unlabeled data points for future labeling in order to maximize accuracy while reducing the amount of input provided by the user.

### 2.3. Inconsistencies in the evaluation protocol

In the literature does not exist a proper benchmark that integrates few-shot and continual learning. Some related tasks were hastily introduced as a mean to prove the efficacy of a given system, making such tasks very restricted in terms of what methods they are applicable on and how many aspects they can investigate. We found that tasks and datasets vary from paper to paper, making it challenging to know the actual performance of a given algorithm. For instance, the method proposed by Vuorio et al. (2018), an extention of MAML able to act as a loss function in the inner loop

of the algorithm, has been tested exclusively on variants of MNIST. The method of Javed & White (2019), an online meta-objective that minimize catastrophic forgetting, has been tested on Omniglot and incremental sine-waves. The work of Finn et al. (2019), another extension of MAML to the online setting, has been evaluated on MNIST, CIFAR-100 and PASCAL 3D+. These inconsistencies in the evaluation protocol of continual few-shot algorithms further support our proposal of a unified benchmark.

Related to continual few-shot learning is the field of *incremental few-shot learning* (Qiao et al., 2018; Gidaris & Komodakis, 2018). The difference between the two lies in how the target sets are sampled during the evaluation phase. In incremental few-shot learning the end performance of trained models is evaluated on target sets sampled from classes encountered at meta-training phase as well as new classes sampled from the evaluation dataset. In continual few-shot learning, during evaluation, support and target sets are sampled only from the test set. Incremental and continual few-shot learning are tangential, the two share similar objectives but are significantly different in terms of training and testing procedures. For this reason we will not analyze this line of research any further.

In conclusion, from this literature analysis it is evident how the problem of continual few-shot learning is not well defined, making it challenging to benchmark and compare performance of algorithms. In the next section, we will focus on formalizing the problem and then we will propose a unified set of tasks and datasets to encourage consistent benchmarking.

## 3. Continual Few-Shot Learning [2]

This section contains the core contribution of the article. We divide the section in three parts: definition of the problem, where we present a principled formulation of continual few-shot learning; definition of the procedure, where we detail the type of tasks that can be used for learning; definition of the dataset, where we describe the desiderata of a suitable dataset and introduce SlimageNet64.

### 3.1. Definition of the problem

In standard few-shot learning (FSL) for classification a task consists of a small training set (i.e. a support set) $\mathcal{S} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N_S}$ of input-label pairs, and a small validation set (i.e. a target set) $\mathcal{T} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N_T}$ of previously unseen pairs. To reduce notation burden we assume that each data-point $x$ has been flattened into a vector of dimensionality $H$. Each label $y \in \mathcal{C}$ with $\mathcal{C}$ being a finite set of classes $\mathcal{C} = \{c_n\}_{n=1}^{N_C} \in \mathbb{N}$. Moreover it is assumed that the pairs in the support and target sets, have different inputs $\mathcal{S}^x \cap \mathcal{T}^x = \{\emptyset\}$ but same class set $\mathcal{S}^y = \mathcal{C} \wedge \mathcal{T}^y = \mathcal{C}$, where we have used the shorthand $\mathcal{S}^x = \{\mathbf{x}_n\}_{n=1}^{N_S}$, $\mathcal{S}^y = \{y_n\}_{n=1}^{N_S}$ (likewise for $\mathcal{T}$). The objective of the learner is to perform well on the validation set $\mathcal{T}$ having only access to the labeled data contained in the support $\mathcal{S}$. The size of the support set $N_S$ is defined by the number of classes $N_C$ (way) and by the *number of samples per class* $K$ (shot), such that if we have a 5-way/1-shot setup we end up with $N_S = N_C \times K = 5 \times 1 = 5$.

In a continual few-shot learning (CFSL) task (i.e. an episode) a single support set is replaced by a sequence of support sets $\mathcal{G} = \{\mathcal{S}_n\}_{n=1}^{N_G}$ with the target set $\mathcal{T} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N_T}$ now containing previously unseen instances of classes stemming from $\mathcal{G}$. We will refer to $N_G$, the cardinality of $\mathcal{G}$, as the *Number of Support Sets Per Task (NSS)*. Here, each support set in $\mathcal{G}$ contains $N_S$ input-output pairs and is defined as $\mathcal{S} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N_S}$ like in the standard setup. We also define another parameter, the *Class-Change Interval (CCI)*, that dictates how often the classes should change, in numbers of support sets. This correspond to assign the elements in the support sets to a series of disjoint class sets $\bigcap_{i=1}^{I} \mathcal{C}_i = \{\emptyset\}$. For example, if CCI=2 then we will draw support sets whose classes change every 2 samples. As a result, support sets $\mathcal{S}_1$ and $\mathcal{S}_2$ will contain different instances of the same class set $\mathcal{C}_1$, whereas $\mathcal{S}_3$ and $\mathcal{S}_4$ will contain different instances from the class set $\mathcal{C}_2$. The process of generating CFSL tasks is also described in Algorithm 1 and implemented in the data provider GitHub repository[3].

A *learner* is a process which extracts task-specific information and distills it into a classification model. The model can be generically defined as a function $f(\mathbf{x}, \boldsymbol{\theta})$ parameterized by a vector of weights $\boldsymbol{\theta}$. At evaluation time the learner is tested through a loss function

$$\mathcal{L} = \Big(f(\mathbf{x}_\mathcal{T}, \boldsymbol{\theta}), y_\mathcal{T}\Big), \tag{1}$$

where $\mathbf{x}_\mathcal{T}$ and $y_\mathcal{T}$ are the input-output pairs belonging to the target set. Note that we intentionally provided a definition that is generic enough to fit into different methodologies and not restricted to the use of neural networks.

To remove the possibility of converting a continual learning task to a non-continual one, we introduce a restriction, which dictates that a support set $\mathcal{S}$ is sampled from $\mathcal{G}$ without replacement, and deleted once it has been used by the learner. The learner should never have access to more than one support set at a time, and should not be able to review a

[2] A full specification sheet of the proposed setting can be found at https://antreasantoniou.github.io/documents/continual_few_shot_learning_specifiation.pdf

[3] The task generator data provider repository can be found at https://github.com/AntreasAntoniou/FewShotContinualLearningDataProvider


Figure 2: Visual representation of the four continual few-shot task types. Each row corresponds to a task with Number of Support Sets, NSS=4, and a defined Class-Change Cnterval (CCI). Given a sequence of support sets, $\mathcal{S}_n$, the aim is to correctly classify samples in the target set, $\mathcal{T}$. Colored frames correspond to the associated support set labels.

support set once it has moved to the next one. This restriction induces a strict sequentiality in the access of $\mathcal{G}$.

The setup we have described so far is very flexible, and it allows us to define a variety of different tasks and therefore to target different problems. In the following section we provide a description of those tasks and show that they are consistent with the continual learning literature.

**Algorithm 1** Sampling a Continual Few-Shot Task

**Data:** Given labeled dataset $\mathcal{D}$, number of support sets per task $NSS$, number of classes per support set $N_C$, number of samples per support set class $K_S$, number of samples per class for target set $K_T$, class change interval $CCI$, and class overwrite parameter $O$

$a = 1, b = 1$
**for** $a \leq (NSS/CCI)$ **do**
  Sample and remove $N_C$ classes from $\mathcal{D}$
  **for** $b \leq CCI$ **do**
    $n \leftarrow a \times CCI + b$
    Sample $K_S + K_T$ samples for each of $N_C$ classes
    Build support $\mathcal{S}_n$ with $K_S$ samples per class
    Build target $\mathcal{T}_n$ with $K_T$ samples per class
    **if** $O = TRUE$ **then**
      Assign labels $\{1, \ldots, N_C\}$ to the classes
    **else**
      Assign labels $\{1 + (a-1) \times N_C, \ldots, N_C \times a\}$ to the classes
    **end**
    Store sets $\mathcal{S}_n$ and $\mathcal{T}_n$
  **end**
**end**
Combine all target sets $\mathcal{T} = \bigcup_{n=1}^{N_G} \mathcal{T}_n$
Return $(\mathcal{S}_{1 \ldots N_G}, \mathcal{T})$

### 3.2. Task Types [4]

In the previous section we have defined the theoretical framework of CFSL, here instead we define an empirical procedure under the form of specific task types. To do so we refer to the literature on continual learning, which has recently focused on more structured procedures, without reinventing the wheel. Note that this is not straightforward, since it is necessary to align the continual learning definitions with the few-shot ones. In continual learning, there are three generally-accepted scenarios in the context of object recognition (Parisi et al., 2019; Lomonaco & Maltoni, 2017): New Instance (NI), New Class (NC) and New Instance and Class (NIC).

In NI, new patterns of a known set of classes become available with each data batch in a sequence. In NC, new classes become incrementally available. The NIC generalises both types of tasks and incrementally releases patterns of known and new sets of classes.

Our categorization of CFSL fully covers the standard continual learning setting while introducing an additional, super-class NI setting. Specifically, task A and B are equivalent to NI and NC, respectively. Task C captures the super-set NI setting where instances are sampled across super-classes, instead of being sampled from previously defined class categories. Finally, task D explores the NIC setting. Figure 2 showcases a high-level visual representation of the proposed task.

A **New Samples:**

**Definition:** In this task type, support sets within a given task are sampled from the a preselected set of classes. As a result, any given support set within a task will share the same classes with all other support sets within that task, but will have previously unseen

[4] For a full implementation of a task generator data provider see Footnote 3

Table 1: Dataset comparisons. Dataset details include: number of classes in the whole dataset (**# Classes**), number of samples per class (**# Samples**), total number of images (**# Total**), **Resolution**, **Format**, and finally, **Size** indicating the allocation of RAM for the whole dataset. Suitability include: class diversity (**Diversity**), enough classes (**# Classes**), enough samples (**# Samples**), proper size (**Size**). Omniglot and SlimageNet64 are the best choices for the tasks on grayscale and RGB datasets, respecitively, according to our suitability criteria (for details see section 3.4).

| | Dataset details | | | | | | Suitability (satisfies criteria) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | **# Classes** | **# Samples** | **# Total** | **Resolution** | **Format** | **Size (GB)** | **Diversity** | **# Classes** | **# Samples** | **Size** |
| MNIST (LeCun, 1998) | 10 | 7000 | 70k | 28×28 | Grayscale | ∼0.20 | ✗ | ✗ | ✗ | ✓ |
| Fashion MNIST (Xiao et al., 2017) | 10 | 7000 | 70k | 28×28 | Grayscale | ∼0.20 | ✗ | ✗ | ✗ | ✓ |
| **Omniglot** (Lake et al., 2015) | 1622 | 20 | ∼32.4k | 28×28 | Grayscale | ∼0.095 | ✓ | ✓ | ✓ | ✓ |
| CUB-200 (Welinder et al., 2010) | 200 | 20-39 | 6033 | ∼475× ∼400 | RGB | ∼13 | ✗ | ✗ | ✗ | ✓ |
| Mini-ImageNet (Vinyals et al., 2016) | 100 | 600 | 60k | 84×84 | RGB | ∼4.7 | ✗ | ✗ | ✓ | ✓ |
| Tiered-ImageNet (Ren et al., 2018) | 608 | 600 | ∼365k | 84×84 | RGB | ∼29 | ✓ | ✓ | ✓ | ✗ |
| CIFAR-100 (Krizhevsky et al., 2009) | 100 | 600 | 60k | 32×32 | RGB | ∼0.68 | ✗ | ✗ | ✓ | ✓ |
| CORe50 (Lomonaco & Maltoni, 2017) | 10 | ∼16.5k | ∼165k | 128×128 | RGB-D | ∼30 | ✗ | ✗ | ✗ | ✗ |
| ILSVRC2012 (Russakovsky et al., 2015) | 1000 | 732-1300 | ∼1.43M | 224×224 | RGB | ∼800 | ✓ | ✓ | ✗ | ✗ |
| **SlimageNet64 (ours)** | 1000 | 200 | 200k | 64×64 | RGB | ∼9.1 | ✓ | ✓ | ✓ | ✓ |

instances (i.e. samples) of those classes:

$$\forall \mathcal{S}_i, \mathcal{S}_j \in \mathcal{G}(\mathcal{S}_i^x \cap \mathcal{S}_j^x = \{\emptyset\} \wedge \mathcal{S}_i^y = \mathcal{S}_j^y = \mathcal{C}), \quad (2)$$

where we have assumed that $\mathcal{S}_i \neq \mathcal{S}_j$. For example, if we have 5 classes per support set and 10 support sets, then by the end of the task we have seen 5 classes, each with 10 samples. To achieve this, we can set CCI to be equal to the number of support sets in a given task (CCI = NSS), which means that for every support set we sample new instances and the same classes (as in previous support sets of the same task).

**Motivation:** Since this setting emulates the default deep learning training regime, it can be useful in studying mini-batch stochastic optimization models as well as meta-learning more efficient algorithms for doing so. It can also be useful when such processes must be executed on a robotic or embedded system.

B **New Classes:**

**Definition:** In this task type, each support set has different classes from the other support sets within a given task, formally we write:

$$\forall \mathcal{S}_i, \mathcal{S}_j \in \mathcal{G}(\mathcal{S}_i^x \cap \mathcal{S}_j^x = \{\emptyset\} \wedge \mathcal{S}_i^y \cap \mathcal{S}_j^y = \{\emptyset\}), \quad (3)$$

with $\mathcal{S}_i \neq \mathcal{S}_j$. Similarly to the previous task, here we focus on the case where every class has just a single associate input $\mathbf{x}$ (1-shot). In this task each class within each support set has a corresponding unique output unit in the model. For example, if each support set contains 5 classes and we have 10 support sets, the model will have a total of 50 output units, one for each class. To achieve this, we set CCI to 1, which means that for every task we sample new classes.

**Motivation:** This setting emulates standard continual learning, where new concepts/classes are acquired as the agent receives a data stream. Therefore it is very useful as a means to investigate such settings or meta-learn models that do well on it. Since this setting allows expanding the number of class descriptors, it is not forced to explicitly rewrite previous knowledge at the class-level, however, it almost always will be required to rewrite representations at lower-levels.

C **New Classes with Overwrite:**

**Definition:** This task is identical to the previous one in terms of how support set inputs are sampled.

The only difference is that the true labels in each support set are *overwritten* by new labels in $C$. This is achieved using the surjective function $O : y \mapsto \widetilde{y}$ that takes as input the labels of a support set $\mathcal{S}^y$ and a class set $\widetilde{\mathcal{C}}$, and returns a new support set $\widetilde{\mathcal{S}} = \{(\mathbf{x}_n, \widetilde{y}_n)\}_{n=1}^{N_S}$, with $\widetilde{y} \in \widetilde{\mathcal{C}}$. We can formally write this as:

$$\forall \mathcal{S}_i, \mathcal{S}_j \in \mathcal{G}(\mathcal{S}_i^x \cap \mathcal{S}_j^x = \{\emptyset\} \wedge \mathcal{S}_i^y \cap \mathcal{S}_j^y = \{\emptyset\} \wedge O(\mathcal{S}_i^y, \widetilde{C}) = O(\mathcal{S}_j^y, \widetilde{C}) = \widetilde{\mathcal{S}}_i^y = \widetilde{\mathcal{S}}_j^y = \widetilde{C}), \quad (4)$$

where $\mathcal{S}_i \neq \mathcal{S}_j$. This task is similar to task A in terms of the number of output units, however, in task C a single output unit is associated with more than one true class. Intuitively, $\widetilde{\mathcal{C}}$ could be the hierarchical categories of classes in $\mathcal{G}^y = \cup_{n=1}^{N_G} \mathcal{S}_n^y$, however, we assign the hierarchical categories arbitrarily.

In practical terms, if we have 5 classes and 10 support sets, in this task the model only uses 5 output units to store all 50 classes. Therefore, for every support set the output unit of a specific class is overwritten with a new one. To obtain this task we need to set CCI to 1, then apply the overwrite function.

**Motivation:** This setting emulates situations where an agent is tasked with learning data-streams while being limited in storing that knowledge in a preset number of output classification labels. As a result the agent learns super classes. This setting is useful in investigating how effective a learner is in continually updating

a class descriptor while not forgetting previous descriptions. Since this setting does not allow expanding the number of class descriptors, it is forced to explicitly rewrite previous knowledge at the class-level, with which certain types of models might struggle more than others. This setting is especially useful for robotics and embedded system applications.

D **New Classes with New Samples:**

**Definition:** In this task type, the sampled support sets contain different instances of the same classes for some predefined CCI ($1 < \text{CCI} < \text{NSS}$) such that

$$\forall \mathcal{S}_i, \mathcal{S}_j \in \mathcal{G}(\mathcal{S}_i^y = \mathcal{S}_j^y \leftrightarrow \mathcal{S}_i \in \mathcal{G}_k \wedge \mathcal{S}_j \in \mathcal{G}_k), \quad (5)$$

where $\mathcal{G}_k$ is a partition of the task set $\mathcal{G}$ satisfying

$$|\mathcal{G}_k| = \text{CCI}, \quad \bigcap_{k=1}^{N_G/\text{CCI}} \mathcal{G}_k = \{\emptyset\}, \quad \bigcup_{k=1}^{N_G/\text{CCI}} \mathcal{G}_k = \mathcal{G}. \quad (6)$$

Note that this partitioning ensures that the subsets are pairwise disjoint. If we have 5 classes per support set, 10 support sets and a CCI of 5, we end up with 5 support sets containing samples from 5 classes and other 5 support sets containing samples from 5 different classes. This makes a total of 10 classes, each one containing 5 samples.

**Motivation:** This setting emulates situations where an agent is tasked with both learning new class descriptors and updating such descriptors by observing new class instances. This setting sheds light on how agents can perform on a setting that mixes all previous settings into one.

### 3.3. Metrics

In this section we provide a number of metrics useful in comparing different models applied to the CSFL setting. It is important to note that each one of this metrics only provides a quantifier for a desirable property. Whether a model is superior to another can only be said when comparing them on the same metric. Whether a model is more desirable than another depends on the task and hardware that a system is trying to solve.

#### 3.3.1. Test Generalization Performance

A proposed model should be evaluated on at least the test sets of Omniglot and SlimageNet, on all the tasks of interest. This is done by presenting the model with a number of previously unseen continual tasks sampled from these test sets, and then using the target set metrics as the task-level generalization metrics. To obtain a measure of generalization across the whole test set the model should be evaluated on a number of previously unseen and unique tasks. The mean and standard deviation of both accuracy and performance should be used as generalization measures to compare models.

#### 3.3.2. Across Task Memory (ATM)

Even though we have imposed a restriction on the access to $\mathcal{G}$, the learner is still authorized to store in a local *memory bank* $\mathcal{M}$ some representations of the inputs and/or output vectors (often implemented as embedding vectors or inner loop parameters)

$$\mathcal{M} = \{(\hat{\mathbf{x}}, \hat{y})_{\mathcal{S}_1}, ..., (\hat{\mathbf{x}}, \hat{y})_{\mathcal{S}_{N_G}}\}, \quad (7)$$

where $\hat{\mathbf{x}}$ and $\hat{y}$ are representations of $\mathbf{x}$ and $y$ obtained after a given learner has processed $\mathbf{x}$ and $y$ and stored some of their useful components. Most learners will be compressing a given support set, but this is not strictly the case.

Note that the potential compression rate is not directly correlated to the complexity of the model (e.g. number of parameters, FLOPs, etc). For instance, compression can be achieved by removing some of the dimensions of the input, or by using a lossless data compression algorithm, which may not require additional parameters or may have minimal impact on the execution time. In this regard, the concept of memory bank $\mathcal{M}$ helps to disambiguate model complexity from any additional memory allocated for compressed representations of inputs. We can use the cardinality of $\mathcal{M}$, indicated as $|\mathcal{M}|$, to quantify the learner efficiency. Given two learners with their corresponding models $f(\mathbf{x}, \boldsymbol{\theta}_1)$ and $f(\mathbf{x}, \boldsymbol{\theta}_2)$, and assuming that the size of $\boldsymbol{\theta}_1$ is equal to the size of $\boldsymbol{\theta}_2$ with $\mathcal{L}_1 = \mathcal{L}_2$, then the learner with smaller cardinality $|\mathcal{M}|$ must be preferred.

In order to compare performances across different tasks and datasets, we relate the size of the stored task-specific representations (in bytes) $\mathcal{M}^{\hat{x}}$ (e.g. embedding vectors in ProtoNets, and inner loop parameters for MAML) during task-specific information extraction to the size of vectors (in bytes) $\mathbf{x}$ contained in the episode $\mathcal{G}^x = \cup_{n=1}^{N_G} \mathcal{S}_n^x$. Recall that $\hat{\mathbf{x}}$ is a compressed version of $\mathbf{x}$ and therefore $F < H$. To reduce the notation burden we have only considered the inputs $\mathbf{x}$ and not the targets $y$, since $\mathbf{x}$ is significantly larger than $y$. Based on these considerations we define Across-Task Memory (ATM)

$$\text{ATM} = \frac{|\mathcal{M}^{\hat{x}}|}{|\mathcal{G}^x|}, \quad (8)$$

where $\mathcal{M}^{\hat{x}}$ is the stored representation of a series of support sets and $\mathcal{G}^x$ is the size of the support sets. Note that for each utilized floating point arithmetic unit we include a computation that takes into account the floating point precision level. For example, if both $\mathcal{M}^{\hat{x}}$ and $\mathcal{G}^x$ use the same floating point standard then it is divided out, but if the representational

form uses a lower precision than the actual data-points then it becomes compressive. From a practical standpoint (image classification), the ATM can be estimated relating the total number of bytes stored in the memory bank (ATM numerator) with the total number of bytes over all the images in the episode (ATM denominator). Given the definition above: ATM $< 1$ for learners with efficient memory, ATM $= 0$ for learners with no memory, and ATM $> 1$ for learners with inefficient memory. Note that the ATM is undefined for empty episodes $\mathcal{G} = \{\emptyset\}$. ATM is task/dataset agnostic and can be used to compare various models (or the same model) across different settings.

To summarize ATM is useful for the following reasons:

1. We do not restrict our agents to a specific amount of memory for their continual task learning. As a result, an agent could easily store whole support sets into its memory bank. We want to be able to distinguish between more memory efficient models (that might in compress support sets efficiently) and less memory efficient models.
2. Using default measures of computational capacity such as MACs is not enough. MACs do not quantify the actual memory shared across the learning process, but instead quantifies the overall computational requirements of the models. Such memory requirements might be minuscule when compared to the model architecture functions which are usually orders of magnitude more expensive. Therefore there is a need for a quantifier that focuses on the efficiency of the learner at compressing incoming data, and how that varies with additional number of support sets.

#### 3.3.3. Multiply-Addition operations (MACs)

This metric measures the computational expense of both the learner and the model operations during learning and inference time. This is different than ATM, as ATM reflects how much memory is required to store information about a support when the next support set is observed, whereas the inference memory footprint measures the memory footprint that the model itself needs to execute during one cycle of inference, and meta-learning cycle.

#### 3.3.4. FSL vs CFSL vs CL

At this point, it is important to properly explain what the relationship between FSL, CSFL and CL is. We argue that all three belong in a spectrum within which the free variables are size of an incoming support set, and the number of support sets within a task. If the size of a support set is very small, e.g. five samples consisting of a single sample from five classes and the number of support sets is one, then we have few-shot learning. If we increase the number of support sets to more than one up to a hundred steps, we have CFSL. Once we begin to increase the size of the support set to something reminiscent of standard deep learning training (e.g. within the range of 32-256 where most models are trained) and we increase the number of support sets into the thousands, we end up with the full continual learning setting.

### 3.4. Datasets

Properly training and evaluating a CFSL agent can be an arduous process. Building such tasks requires datasets that meet the following desiderata:

1. **Diversity:** An optimal dataset should have a very high degree of diversity in terms of classes. This enforces robustness in the learning procedure, since the model has to be able to deal with previously unseen class semantics. In addition, diversity enable the training, validation, and test splits to lie within different distribution spaces, covering classes that are significantly different from one another.
2. **Number of classes:** The dataset should contain a very high number of categories. This is to ensure that we can train models on CFSL tasks ranging from 1 sub-task, all the way to 100s of sub-tasks. Ideally, the length of a sub-task sequence should not be constrained by the number of classes in the dataset.
3. **Number of samples per class:** The dataset should contain a fair, but not overabundant, number of samples per class. On the one hand, a dataset with few samples can not capture the difference in distribution within each class, resulting in a poor evaluation measure. Moreover, training a learner on a small dataset can produce significant underfitting issues. On the other hand, having too many samples per class increases the training time, producing very strong learners but neutralizing the difference among them. As a result, it would be much harder to draw any conclusion on the capabilities of the underlying algorithms, since the difference in performance between them would be minimal.
4. **Size:** Finally, we would like our models to be trained in reasonable time, finances and computational resources. Thus, the size of the dataset should be contained, such that it can be easily managed and stored in memory. This requirement is crucial to allow use of the dataset by a significant portion of the research community. Here, we define a dataset as appropriate if its size does not exceed 16 GB, which is our reasonable estimate of the average laptop RAM.

Table 2: Accuracy and standard deviation (percentage) on the test set for the proposed benchmarks and tasks. Best results in bold.

| | Task Type | FSL | B | C | A | D | B | C | A | D | B | C | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NSS | 1 | 3 | 3 | 3 | 4 | 5 | 5 | 5 | 8 | 10 | 10 | 10 |
| | CCI | 1 | 1 | 1 | 3 | 2 | 1 | 1 | 5 | 2 | 1 | 1 | 10 |
| | Overwrite | - | False | True | True | False | False | True | True | False | False | True | True |
| Omniglot | Init + Tune | 43.05±0.01 | 10.87±0.01 | 27.51±0.01 | 44.76±0.01 | 8.74±0.01 | 6.15±0.01 | 24.52±0.01 | 45.30±0.01 | 3.93±0.01 | 3.12±0.01 | 22.16±0.01 | 45.64±0.01 |
| Omniglot | Pretrain + Tune | 33.07±2.04 | 9.97±0.14 | 26.75±0.27 | 32.44±0.29 | 7.91±0.15 | 6.02±0.02 | 24.51±0.06 | 31.89±1.10 | 3.86±0.06 | 3.13±0.03 | 22.30±0.06 | 33.17±0.39 |
| Omniglot | ProtoNets | 98.52±0.04 | 95.30±0.12 | 45.44±0.19 | 98.73±0.02 | 48.98±0.03 | 91.52±0.20 | 35.10±0.09 | 98.73±0.12 | 48.44±0.03 | 83.72±0.19 | 27.39±0.17 | 98.65±0.14 |
| Omniglot | MAML++ L | 99.46±0.03 | 38.18±0.14 | 46.12±0.15 | 99.38±0.07 | 28.87±0.07 | 22.69±0.07 | 35.76±0.14 | 99.41±0.04 | 14.29±0.05 | 11.30±0.02 | 27.82±0.03 | 99.44±0.01 |
| Omniglot | MAML++ H | 99.54±0.03 | 96.14±0.02 | 96.77±0.08 | 99.73±0.04 | 49.44±0.02 | 92.70±0.03 | 93.47±0.05 | 99.80±0.01 | 49.00±0.04 | 85.56±0.10 | 86.38±0.14 | 99.86±0.01 |
| Omniglot | SCA | **99.78±0.01** | **96.84±0.04** | **97.38±0.02** | **99.82±0.01** | **49.71±0.01** | **93.81±0.02** | **94.08±0.45** | **99.88±0.03** | **49.51±0.01** | **86.07±0.03** | **87.29±0.19** | **99.88±0.01** |
| SlimageNet64 | Init + Tune | 25.1±0.01 | 8.4±0.01 | 21.3±0.01 | 24.4±0.01 | 6.1±0.01 | 4.5±0.01 | 20.8±0.01 | 24.7±0.01 | 3.0±0.01 | 2.4±0.01 | 20.5±0.01 | 24.9±0.01 |
| SlimageNet64 | Pretrain + Tune | 24.5±0.60 | 8.7±0.03 | 21.9±0.11 | 24.2±0.17 | 6.4±0.01 | 4.9±0.02 | 21.2±0.05 | 24.5±0.23 | 3.3±0.03 | 2.7±0.03 | 20.7±0.10 | 24.4±0.20 |
| SlimageNet64 | ProtoNets | 41.8±0.16 | 24.1±0.05 | 25.9±0.23 | 43.1±0.24 | 15.1±0.03 | 18.2±0.14 | 22.7±0.09 | 43.3±0.03 | 10.4±0.12 | 12.3±0.09 | 21.0±0.06 | 43.7±0.15 |
| SlimageNet64 | MAML++ L | 42.0±0.48 | 13.6±0.04 | 25.5±0.23 | 42.7±0.10 | 10.2±0.11 | 7.9±0.13 | 22.6±0.03 | 43.0±0.12 | 5.0±0.08 | 3.6±0.14 | 20.8±0.09 | 43.0±0.42 |
| SlimageNet64 | MAML++ H | 45.3±0.14 | 27.2±0.25 | 33.8±0.16 | 61.2±0.36 | 16.8±0.18 | 21.0±0.21 | **30.4±0.51** | 68.6±0.47 | 12.3±0.11 | 14.4±0.12 | 25.7±0.10 | 75.6±0.10 |
| SlimageNet64 | SCA | **46.6±0.16** | **27.9±0.16** | **34.0±0.23** | **65.3±0.15** | **17.3±0.07** | **22.0±0.18** | 30.1±0.36 | **72.0±0.36** | **12.7±0.08** | **14.6±0.07** | **26.3±0.13** | **77.4±0.06** |

Many datasets already exist in continual and few-shot learning, however most of them do not satisfy all the aforementioned requisites and are insufficient for robust benchmarking of CFSL algorithms. Omniglot (Lake et al., 2015) was a good first choice for a lower-difficulty dataset, however, we were still missing a higher complexity dataset with coloured images.

For this reason we propose a new variant of ImageNet64×64 (Chrabaszcz et al., 2017), named *SlimageNet64* (derived from Slim and ImageNet). SlimageNet64 consists of 200 instances from each of the 1000 object categories of the ILSVRC-2012 dataset (Krizhevsky et al., 2012; Russakovsky et al., 2015), for a total of 200K RGB images with a resolution of $64 \times 64 \times 3$ pixels. We created this dataset from the downscaled version of ILSVRC-2012, ImageNet64x64, as reported in (Chrabaszcz et al., 2017), using the *box* downsampling method available from *Pillow* library. In Table 1 we report a detailed comparison of all the datasets available, showing how SlimageNet64 is an optimal choice in terms of diversity, number of classes, number of samples per class, and storage size. The closest alternative to SlimageNet64 is Tiered-ImageNet (Ren et al., 2018), a subset of ILSVRC-12 with a total of 608 classes. Comparing the two, SlimageNet64 contains more classes and overall has a higher class diversity across train, validation, and test. Moreover, it has a lower computational footprint due to the smaller resolution of the images and the lower number of samples per class. These characteristics make SlimageNet64 more compact and at the same time more challenging.

## 4. Experiments [5]

For the purposes of establishing baselines in the CFSL tasks outlined in this paper we chose to use six existing FSL methods: (i) randomly initializing a convolutional neural network, and fine tuning on incoming tasks, (ii) pretraining a convolutional neural network on all training set classes and then fine-tune on sequential tasks (Chen et al., 2019), (iii) Prototypical Networks (Snell et al., 2017) (baseline for metric-based FSL methods), (iv) the Improved Model Agnostic Meta-Learning or MAML++ L (Low-End) model (Antoniou et al., 2019) (baseline for optimization based FSL methods), (v) MAML++ H (High-End) model (Antoniou & Storkey, 2019) (dense-net backbone, squeeze excite attention, mid-tier baseline), and (vi) the Self-Critique and Adapt model (SCA) (Antoniou & Storkey, 2019), a top state-of-the-art algorithm for FSL (high-tier baseline). For each model, we used the exact configurations specified in their original papers. For each method (apart from ProtoNets) we used five inner-loop update steps.

For each continual learning task type, we ran experiments on each dataset. Each support set contained 1 sample from 5 classes (5-way, 1-shot) while the target sets contained 5 samples from all the classes seen in a given task. We ran experiments using 1, 3, 5 and 10 support sets for each continual task, therefore creating tasks of increasingly long number of sub-tasks. We ran each experiment 3 times, each time with different seeds for the data-provider and the model initializer. All models were trained for 250 epochs, where each epoch consisted of 500 update steps, each one done on a single continual task, using the default configuration of the Adam learning rule, and weight-decay of 1e-5. At the end of each training epoch we validated a given model by applying it on 600 randomly sampled continual tasks, keeping those tasks consistent across all validation phases. Once all epochs have been completed, we built an ensemble of the top five models across all epochs with respect to validation accuracy, and applied that on 600 random tasks sampled from the test set, to compute the final performance metrics.

For Omniglot, we used the first 1200 classes for the training set, and we split the rest equally to create a validation and test set. For SlimageNet64, we used 700, 100 and 200 classes to build our training, validation and test sets respectively. The SlimageNet64 splits were chosen such that the training set had mostly living organisms, with some additional everyday tools and buildings, while the validation

[5] We provide an implemetation that reproduces all the experiments in this section at https://github.com/AntreasAntoniou/FewShotContinualLearning

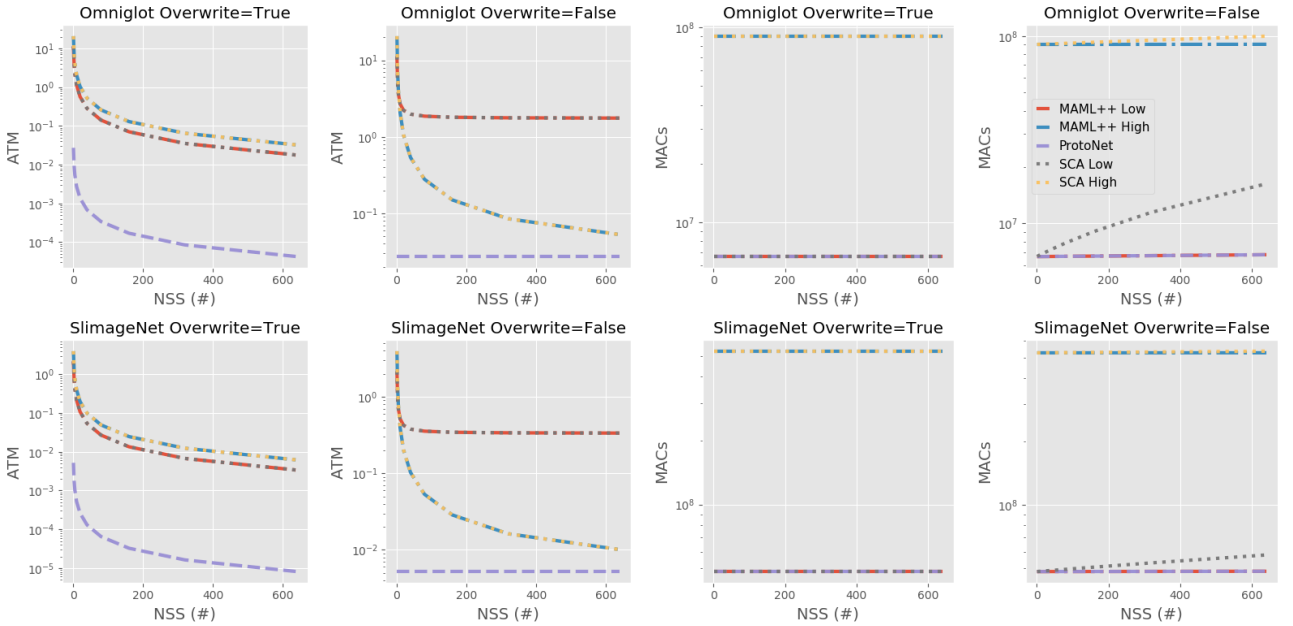


Figure 3: ATM (Across-Task Memory) and MAC (Multiply-Accumulate Computations) costs for a variety of NSS (Number of Support Sets Per Task). ProtoNets are the superior method across the board. In terms of ATM it is worth noting that methods such as MAML++ H and SCA tend to become incrementally cheaper than MAML++ L as the number of support sets increases. Whereas in terms of MACs MAML++ H and SCA are the most expensive by an order of magnitude or more compared to MAML++ L and ProtoNets.

and test sets contained largely inanimate objects. This was done to ensure sufficient domain-shift between the training and evaluation distributions. As a result this enables a more robust generalization measure to be computed.

## 5. Baseline Results

Results are reported in Table 2 and Figure 4. The results from our proposed benchmark, have revealed previously unknown weaknesses and strengths of existing few-shot learning methods. In Omniglot, in the New Classes without Overwrite Setting (B) MAML++ Low-End is inferior to ProtoNets, whilst in the New Classes with Overwrite Settings (C) this result is reversed. From this we can infer that embedding-based methods are better at retaining information from previously seen classes, assuming that each new class remains distinct. However, when overwriting is enabled this trend is overturned because ProtoNet prototypes are shared by a number of super-classes containing classes that are harder to semantically disentangle. Gradient based methods such as MAML++ dominate in this setting, since they can update their weights towards new tasks, and therefore achieve a better disentanglement of those super-classes. SCA and High-End MAML++ (which utilize both embeddings and gradient-based optimization) produce the best performance across all settings. In the New Samples Setting (A), gradient based methods tend to outperform embedding-based methods while hybrid methods produce the best results. Furthermore, in the New Classes and Samples Setting (D), embedding-based methods outperform gradient-based methods, whilst hybrid methods continue to produce the best performing models.

In SlimageNet, ProtoNets seem to consistently outperform the Low-End MAML++ model, even in the New Classes with Overwrite Settings (C) where it was previously inferior. This might indicate that in SlimageNet retaining information about previously seen tasks is more important than disentangling complicated super-classes. Overall models that use both embedding-based and gradient-based methods, seem to outperform methods that do just one of the two often with a performance boost of 100-200%. In the New Classes and Samples Setting (D), embedding-based methods outperform gradient-based ones by a significant margin, while hybrid approaches consistently generate the best performing models. Interestingly, in the New Samples Setting (A) using SlimageNet64, the embedding-based and gradient-based methods produce very similar results to one another, whereas in Omniglot gradient-based methods dominate.

Furthermore Figure 3 shows the ATM and MAC costs for a range of NSS, starting from one, up to and including 640. Some notable observations include the fact that ProtoNets are simply the most efficient in both metrics, by two orders magnitude. In addition, even though the Low-End MAML++ starts off cheaper than the high end model, as NSS increases, it eventually becomes far more expensive than the High-End variant. This is mostly due to the fact

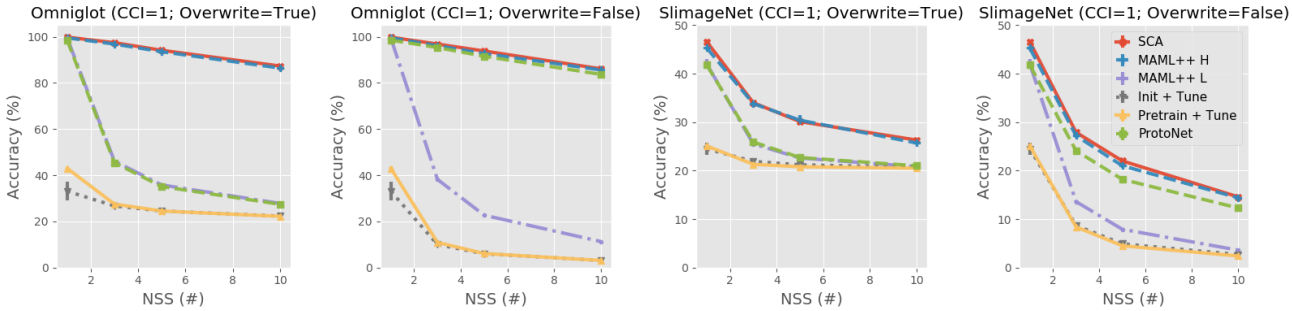


Figure 4: Accuracy (percentage) of different methods on the Omniglot and SlimageNet datasets for different values of Number of Support Sets Per Task (NSS). We report both with/without overwrite. This figure illustrates which methods tend to be more robust to increasing NSS (SCA, MAML ++ H) and which methods do not (ProtoNets, MAML++ L, Init/Pretrain + Tune), as well as to how sensitive they are to those changes.

that the Low-End MAML++ flattens its features and applies a linear layer at the output side of the network. As a result, for each additional new class to be learned, there is one magnitude higher cost than the high-end MAML which simply global pools its features before applying a linear layer.

## 6. Conclusion

In this paper, we have introduced a new flexible and extensive benchmark for Continual Few-shot Learning. We have also introduced a new minimal variant of ImageNet, called SlimageNet64, that contains all of ImageNet classes, but only 200 samples from each class, downscaled to 64×64. The dataset requires just 9 GB of RAM, and it can be easily loaded in memory for faster experimentation. Furthermore, we have run experiments on the proposed benchmarks, utilizing a number of popular few-shot learning models and baselines. In doing so, we have found that embedding-based models tend to perform better when incoming tasks contain different classes from one another, potentially due to better task-specific information retention. On the other hand, gradient-based methods tend to perform better when the task-classes form super-classes of randomly combined classes, resulting in a disentangled task that is harder to predict. Gradient-based methods work better here thanks to their ability of dynamic adaptation, whereas more static methods like ProtoNets tend to produce poorer performances. That being said, in datasets of higher class diversity and sample complexity, gradient-based methods perform like embedding-based methods. We assume that this is due to the nature of the data, making class-information retention more relevant than disentanglement factors. Methods utilizing both embedding-based and gradient-based methods (i.e. High-End MAML++ and SCA) outperform methods that use either of the two. In conclusion, we hope that the proposed benchmark and dataset, will help increasing the rate of progress and the understanding of the behavior of systems trained in a continual and data-limited setting.

## 7. Acknowledgements

We would like to thank Elliot Crowley, Paul Micaelli, Eleanor Platt, Ondrej Bohdal, Sen Wang, and Joseph Mellor for reviewing this work and providing useful suggestions/comments. This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (Grant No. EP/L016427/1) and the University of Edinburgh as well as a Huawei DDMPLab Innovation Research Grant. Furthermore, additional funding for the project was provided by a joint grant by the UK Engineering and Physical Sciences Research Council and SeeByte Ltd (Grant No. EP/S515061/1).

## References

Antoniou, A. and Storkey, A. Learning to Learn by Self-Critique. *Neural Information Processing Systems, NeurIPS*, 2019.

Antoniou, A., Storkey, A., and Edwards, H. Data Augmentation Generative Adversarial Networks. *arXiv preprint arXiv:1711.04340*, 2017.

Antoniou, A., Edwards, H., and Storkey, A. How to train your MAML. *In International Conference on Learning Representations*, 2019.

Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C., and Huang, J.-B. A Closer Look at Few-Shot Classification. In *International Conference on Learning Representations*, 2019.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. A Downsampled Variant of Imagenet as an Alternative to the CIFAR datasets. *Computing Research Repository*, 2017.

Edwards, H. and Storkey, A. Towards a neural statistician. In *International Conference on Learning Representations*, 2017.

Finn, C., Abbeel, P., and Levine, S. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv preprint arXiv:1703.03400*, 2017.

Finn, C., Rajeswaran, A., Kakade, S., and Levine, S. Online Meta-Learning. *arXiv preprint arXiv:1902.08438*, 2019.

Gidaris, S. and Komodakis, N. Dynamic Few-Shot Visual Learning without Forgetting. In *Computer Vison and Pattern Recognition*, 2018.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2019.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

He, X. and Jaeger, H. Overcoming Catastrophic Interference using Conceptor-Aided Backpropagation. *International Conference on Learning Representations*, 2018.

Javed, K. and White, M. Meta-learning representations for continual learning. *arXiv preprint arXiv:1905.12588*, 2019.

Kemker, R., McClure, M., Abitino, A., Hayes, T. L., and Kanan, C. Measuring catastrophic forgetting in neural networks. In *Association for the Advancement of Artificial Intelligence*, 2018.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*, 2012.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 2015.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Lee, S. Toward continual learning for conversational agents. *arXiv preprint arXiv:1712.09943*, 2017.

Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. State representation learning for control: An overview. *Neural Networks*, 2018.

Lesort, T., Caselles-Dupré, H., Garcia-Ortiz, M., Stoian, A., and Filliat, D. Generative models from the perspective of continual learning. In *International Joint Conference on Neural Networks*, 2019a.

Lesort, T., Gepperth, A., Stoian, A., and Filliat, D. Marginal replay vs conditional replay for continual learning. In *International Conference on Artificial Neural Networks*. Springer, 2019b.

Lesort, T., Lomonaco, V., Stoian, A., Maltoni, D., Filliat, D., and Díaz-Rodríguez, N. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *arXiv preprint arXiv:1907.00182*, 2019c.

Li, Z., Zhou, F., Chen, F., and Li, H. Meta-SGD: Learning to Learn Quickly for Few Shot Learning. *arXiv preprint arXiv:1707.09835*, 2017.

Lomonaco, V. and Maltoni, D. Core50: A New Dataset and Benchmark for Continuous Object Recognition. *arXiv preprint arXiv:1705.03550*, 2017.

Lopez-Paz, D. and Ranzato, M. Gradient Episodic Memory for Continual Learning. In *Advances in Neural Information Processing Systems*, 2017.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Computer Vison and Pattern Recognition*, 2018.

Mallya, A., Davis, D., and Lazebnik, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision*, 2018.

Maltoni, D. and Lomonaco, V. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 2019.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. Elsevier, 1989.

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., et al. Never-ending learning. *Communications of the ACM*, 2018.

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

Patacchiola, M., Turner, J., Crowley, E. J., and Storkey, A. Deep kernel transfer in gaussian processes for few-shot learning. *arXiv preprint arXiv:1910.05199*, 2019.

Qiao, S., Liu, C., Shen, W., and Yuille, A. L. Few-Shot image recognition by predicting parameters from activations. In *Computer Vison and Pattern Recognition*, 2018.

Ravi, S. and Larochelle, H. Optimization as a model for Few-Shot Learning. In *International Conference On Learning Representations*, 2016.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. ICARL: Incremental Classifier and Representation Learning. In *Computer Vison and Pattern Recognition*, 2017.

Ren, M., Triantafillou, E., Ravi, S., Snell, J., Swersky, K., Tenenbaum, J. B., Larochelle, H., and Zemel, R. S. Meta-Learning for Semi-Supervised Few-Shot Classification. *arXiv preprint arXiv:1803.00676*, 2018.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. Imagenet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*, 2016.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. Meta-Learning with Latent Embedding Optimization. *International Conference On Learning Representations*, 2019.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. A simple neural network module for Relational Reasoning. In *Neural Information Processing Systems*, 2017.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How Does Batch Normalization Help Optimization? *Neural Information Processing Systems*, 2018.

Schmidhuber, J. Evolutionary principles in self-referential learning. *On learning how to learn: The meta-meta-... hook.) Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1987.

Snell, J., Swersky, K., and Zemel, R. Prototypical Networks for Few-Shot Learning. In *Neural Information Processing Systems*, 2017.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching Networks for One Shot Learning. In *Neural Information Processing Systems*, 2016.

Vuorio, R., Cho, D., Kim, D., and Kim, J. Meta Continual Learning. *Computing Research Repository*, 2018.

Wang, Y., Chao, W.-L., Weinberger, K. Q., and van der Maaten, L. SimpleShot: Revisiting Nearest-Neighbor Classification for Few-Shot Learning, 2019.

Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., and Perona, P. Caltech-UCSD birds 200. 2010.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017.

Zenke, F., Poole, B., and Ganguli, S. Continual Learning through Synaptic Intelligence. In *International Conference on Machine Learning*, 2017.