A Implementation details

Schedule. 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. That is, a Conv-4 architecture for ProtoNets, MAML++ Low End, EWC variants as well as init + fine tune and pretrain + fine tune. In all variants except ProtoNets we adapt the full network's weights in the inner loop. In ProtoNets no inner loop optimization is carried out. For MAML++ High End and SCA we use the improved architecture detailed in Antoniou and Storkey (2019), where there is an image embedding based on a densenet that is not optimized in the inner loop, with a small conv network and a single linear layer which are optimized in the inner loop. 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.

Memory Representations. We have computed the ATM for each model in Appendix C; in this section we will state the exact way each model represents its ATM bank. ProtoNets represents its memory as class embeddings, called prototypes which store a mean of all vectors associated with any given class. In Init + Fine Tune, Pretrain + Fine Tune, MAML++ L and EWC variants, the ATM is represented as the updatable set of weights in the inner loop which span the entirety of the Conv-4 architecture used for each model. Finally, for MAML++ H and SCA, the ATM is represented as a single convolutional layer and a single linear layer, which compose the small inner loop updatable network used by the models to adapt to new continual tasks.

Datasets. 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 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.

B Comparison between datasets

We identified four desiderata that a dataset should have in order to be appropriate for CFSL. Note that, it is hard to define quantitative criteria. Here, we provide qualitative criteria that should be considered as generic desiderata therefore subject to a certain degree of arbitrariness.

Table 2: Dataset comparisons. Details details: number of classes in the whole dataset (**#Classes**), number of samples per class (**#Samples**), **Resolution**, **Format**, **Size** allocation of RAM for the whole dataset. Suitability: 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.

	Dataset details					**Suitability (satisfies criteria)**			
Dataset	**#Classes**	**#Samples**	**Resolution**	**Format**	**Size (GB)**	**Diversity**	**#Classes**	**#Samples**	**Size**
MNIST (LeCun, 1998)	10	7000	28×28	Grayscale	∼0.20	✗	✗	✗	✓
Fashion MNIST (Xiao et al., 2017)	10	7000	28×28	Grayscale	∼0.20	✗	✗	✗	✓
Omniglot (Lake et al., 2015)	1622	20	28×28	Grayscale	∼0.095	✓	✓	✓	✓
CUB-200 (Welinder et al., 2010)	200	20-39	∼475× ∼400	RGB	∼13	✗	✗	✗	✓
Mini-ImageNet (Vinyals et al., 2016)	100	600	84×84	RGB	∼4.7	✗	✗	✓	✓
Tiered-ImageNet (Ren et al., 2018b)	608	600	84×84	RGB	∼29	✓	✓	✓	✗
CIFAR-100 (Krizhevsky et al., 2009)	100	600	32×32	RGB	∼0.68	✗	✗	✓	✓
CORe50 (Lomonaco and Maltoni, 2017)	50	∼16.5k	128×128	RGB-D	∼30	✗	✗	✗	✗
ILSVRC2012 (Russakovsky et al., 2015)	1000	732-1300	224×224	RGB	∼800	✓	✓	✗	✗
SlimageNet64 (ours)	1000	200	64×64	RGB	∼9.1	✓	✓	✓	✓

1. **Diversity:** 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. 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:** 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:** fair, but not overabundant, number of samples per class. A dataset with few samples can not capture the difference in distribution within each class (poor evaluation measure), whereas having too many samples per class increases the training time, producing very strong learners but neutralizing the difference among them.
4. **Size:** should be contained. The model should be trained in reasonable time, finances and computational resources. 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.

C Memory cost

We report the memory comparison on various datasets in terms of Multiply-Accumulate Computations (MACs) and Across Task Memory (ATM). Measuring MACs is a way to quantify the inference cost of each method. Mesuring ATM is useful for two reasons. (i) We do not restrict an agent to a specific amount of memory, therefore it could easily store all support sets into its memory bank. ATM distinguish models in terms of memory efficiency. (ii) Default measures of computational capacity such (e.g. MACs) are not enough, since they quantify the overall computational requirements and not the actual memory shared across the learning process. This might be minuscule when compared to the model architecture functions which are usually orders of magnitude more expensive. ATM quantifies the efficiency of the learner at compressing incoming data.

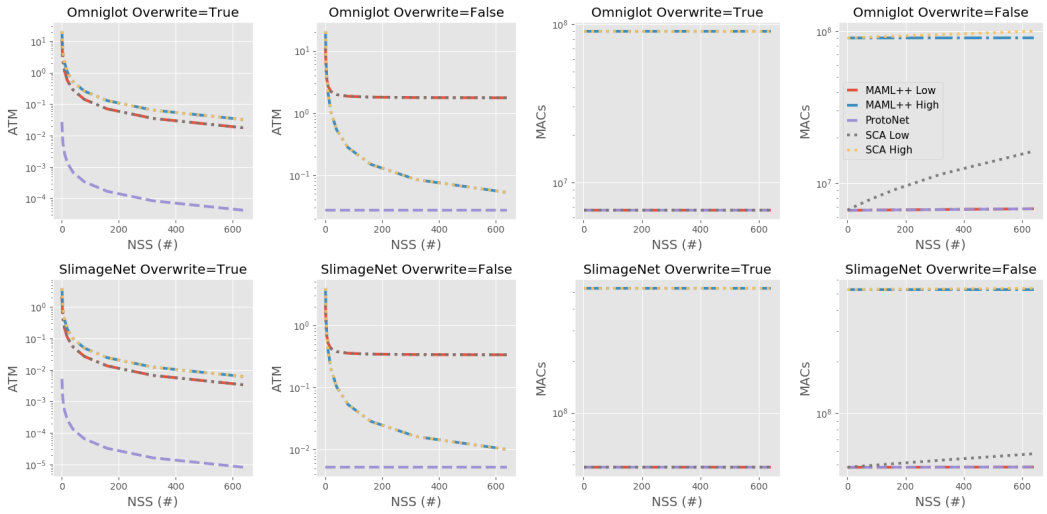

Figure 4: ATM (Across-Task Memory) and MAC (Multiply-Accumulate Computations) costs for a variety of NSS (Number of Support Sets Per Task). ProtoNets are the superior method across the board. In terms of ATM it is worth noting that methods such as MAML++ H and SCA tend to become incrementally cheaper 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.

D Proposed tasks and previous work

We show that our tasks are consistent with the previous work in continual learning (Lomonaco and Maltoni, 2017; van de Ven and Tolias, 2018; Lesort et al., 2019c). Continual learning has

three different scenarios (Lesort et al., 2019c): Single-Incremental-Task (SIT), Multi-Task (MT), Multi Incremental-Task (MIT). The continual learning settings for object recognition fall within the Single-Incremental-Task scenario and can be further partitioned into three update content types (Maltoni and Lomonaco, 2019): New Instances (NI), New Classes (NC), New Instances and New Classes (NIC). In parallel work, van de Ven and Tolias (2018) proposes Non-, Class-, Domain-, and Task- Incremental-Learning (IL) for general continual learning scenarios. This categorization also appears in later work (Hsu et al., 2018; Zeno et al., 2018). We argue that our proposed New-Samples task (A) is most consistent with New Instances (Maltoni and Lomonaco, 2019) but is also similar to Non-IL in (van de Ven and Tolias, 2018) since the distribution of images sampled between support sets does not change between time steps. New-Classes without Overwrite (task B) is most consistent with New-Classes (Maltoni and Lomonaco, 2019) and Class-IL (van de Ven and Tolias, 2018). New-Classes with Overwrite (task C) is not defined in (Maltoni and Lomonaco, 2019) directly but could be seen as a generalization of New Instances where each temporal batch of data (in our case, support sets) is generated from super-classes as opposed to class with a static distribution. New-Classes with Overwrite (task C) is most similar to Domain-IL (van de Ven and Tolias, 2018) where the 'head' dimension (the set of output labels) is kept the same between consecutive batches of data. Finally, our New-Classes with New-Samples task (D) is most similar to New-Classes and Samples (Maltoni and Lomonaco, 2019). This task does not have an equivalent task in (van de Ven and Tolias, 2018). Table 3 shows a disambiguation summary of the continual learning scenarios.

Table 3: Continual Learning task scenarios disambiguation.

	van de Ven and Tolias (2018)			Maltoni and Lomonaco (2019)		
Task	Class-IL	Domain-IL	Task-IL	SIT-NI	SIT-NC	SIT-NIC
NS (A)				✓		
NC w/o O (B)	✓				✓	
NC w/ O (C)		✓				
NCwNS (D)						✓

E Pseudocode

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 $(\mathcal{S}_{1 \ldots N_G}, \mathcal{T})$;