Online Continual Learning for Embedded Devices

Tyler L. Hayes[1], **Christopher Kanan**[1,2]
Rochester Institute of Technology[1], Cornell Tech[2]
{tlh6792, kanan}@rit.edu

Abstract

Real-time on-device continual learning is needed for new applications such as home robots, user personalization on smartphones, and augmented/virtual reality headsets. However, this setting poses unique challenges: embedded devices have limited memory and compute capacity and conventional machine learning models suffer from catastrophic forgetting when updated on non-stationary data streams. While several online continual learning models have been developed, their effectiveness for embedded applications has not been rigorously studied. In this paper, we first identify criteria that online continual learners must meet to effectively perform real-time, on-device learning. We then study the efficacy of several online continual learning methods when used with mobile neural networks. We measure their performance, memory usage, compute requirements, and ability to generalize to out-of-domain inputs[1].

1 Introduction

Continual machine learning systems have the ability to learn from ever-growing data streams (Parisi et al., 2018). In contrast, conventional machine learning algorithms typically assume that there is a static training and evaluation dataset. Continual learning has emerged as a popular research area. One of the most critical applications for continual learning is using it on embedded devices such as mobile phones, virtual/augmented reality (VR/AR) headsets, robots, vehicles, and smart appliances. VR headsets use continual learning to localize the position of the wearer within the boundary that the user has established so that the user does not collide with obstacles (O'Hagan & Williamson, 2020). AR headsets require continual learning to identify relevant objects and regions in the field of view to appropriately position virtual perceptual information. Household robotic devices need to learn the identity of the individuals, pets, and objects in the house. Typically, inference for these applications must be done within embedded devices to minimize latency, but continual on-device learning is critical to preserving privacy and security of the user.

Conventional machine learning systems trained with empirical risk minimization assume that the data is independent and identically distributed (iid), which is typically enforced by shuffling the data. In continual learning, this assumption is violated, which results in catastrophic forgetting (French, 1999; Parisi et al., 2018). Hence, the continual learning research community has focused on solving this catastrophic forgetting problem in a variety of scenarios. However, most of these scenarios do not match the conditions an agent would face for embedded applications. For embedded *supervised* continual learning systems, we argue these capabilities are essential:

1. Online learning and inference in a compute and memory constrained environment.
2. The ability to learn from data in any order without catastrophic forgetting, including learning from iid (shuffled) data streams, streams ordered by category, and everything in between.
3. Making no assumptions about the availability of task labels during inference.
4. Efficiently learning and generalizing with as few labeled examples as possible.

These criteria can be summarized as stating that we need sample efficient, order agnostic, online continual learning (see Fig. 1 for a motivating example).

In this paper, we study continual learning for embedded devices. We make the following contributions:

1. We establish the criteria needed for continual learning on embedded devices, a real-world problem where continual learning is needed.
2. We compare seven algorithms for online learning when used with five convolutional neural networks (CNNs) designed for embedded devices based on their ability to learn from both shuffled data and data sorted by category, which are the extreme best and worst case scenarios, respectively.

[1]https://github.com/tyler-hayes/MAKE-A-REPO

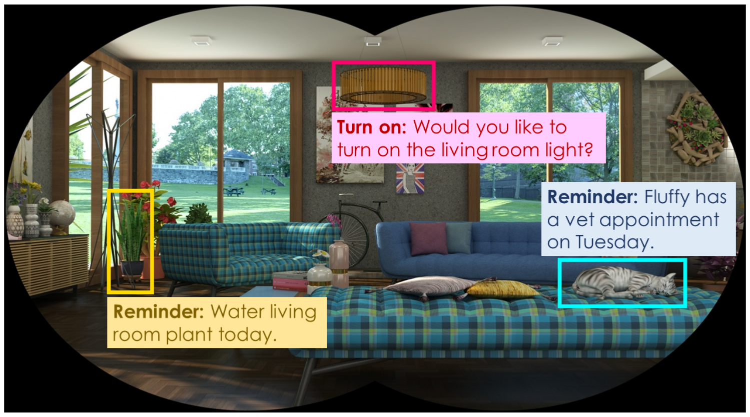

Turn on: Would you like to turn on the living room light?
Reminder: Fluffy has a vet appointment on Tuesday.
Reminder: Water living room plant today.

Figure 1: Using online continual learning, an AR headset could update the locations of specific object instances in the frame of a wearer to provide real-time feedback. This requires immediate learning and inference on the compute and memory constrained headset. Moreover, no assumptions can be made about the order in which the frames of data will be provided and task labels cannot be assumed to be provided. This paper establishes the necessary criteria to support the types of AR applications demonstrated here. Moreover, we also provide initial baselines that meet these criteria.

3. We conduct experiments on three high-resolution image classification datasets: OpenLORIS, Places-365, and Places-Long-Tail. Algorithms are compared based on their ability to learn over time as well as their computational and memory requirements.
4. We make baseline recommendations for future researchers based on each online continual learner's ability to learn from temporally correlated video streams, learn/generalize from very few labeled examples, scale to large-scale data streams with hundreds of classes and millions of examples, and perform well on imbalanced data streams.

2 Problem Formulation

We study *online continual learning* from possibly infinite data streams. That is, a model learns a dataset $\mathcal{D} = \{(\mathbf{X}_t, y_t)\}_{t=1}^{\infty}$ one instance at a time, where $(\mathbf{X}_t, y_t)$ is an example/label pair received at time t. In our paper, all examples are images. Each example is seen only once, where it cannot be revisited unless it is cached in auxiliary memory. The online continual learning setting is needed because many embedded applications receive a limited amount of data and only have a limited amount of time to process the data. For example, a user might use their smartphone to take a handful of photos of objects in their home. A classification model would then be required to rapidly process these few object instances to provide a prediction in real-time. Moreover, many embedded systems collect data in a natural streaming setting (e.g., video cameras collect data frame-by-frame) and it would be ideal to process this data in the real-time order in which it was received.

For online continual learners to be deployed on embedded devices, they must also operate under memory and time constraints to operate on-device. The learner must be performant regardless of the ordering of the instances. This is because an agent operating in the natural world might not have direct control over the order in which visual inputs are received and cannot assume they will be randomly shuffled (iid). Further, agents can only assume access to an input $\mathbf{X}_t$ for inference (i.e., labels indicating which task an input came from may not be available). This is because providing task labels to agents requires additional annotation efforts, which may not be available for some applications. Finally, to learn quickly, agents must be able to generalize from very few examples.

3 Related Work

Continual learning involves training agents over time from non-stationary data streams. However, one challenge with updating agents on non-stationary data distributions is catastrophic forgetting of previous knowledge (McCloskey & Cohen, 1989; French, 1999). To overcome forgetting, researchers have studied regularization techniques (Aljundi et al., 2018; Chaudhry et al., 2018; 2019; Dhar et al., 2019; Kirkpatrick et al., 2017; Li & Hoiem, 2016; Lopez-Paz & Ranzato, 2017; Ritter et al., 2018; Serra et al., 2018; Zenke et al., 2017), network expansion methods (Hayes & Kanan, 2020; Hou et al., 2018; Ostapenko et al., 2019; Rusu et al., 2016; Yoon et al., 2018), and experience replay (Hayes et al., 2021; Belouadah & Popescu, 2019; Castro et al., 2018; Chaudhry et al., 2019; Douillard et al., 2020; French, 1997; Hayes et al., 2019; 2020; Hou et al., 2019; Kemker & Kanan, 2018; Rebuffi et al., 2017; Tao et al., 2020a; Wu et al., 2019). Regularization techniques add a constraint to the loss function to constrain parameter updates. Network expansion methods add new neurons to the network as new tasks become available. Experience replay methods maintain a subset of previous data in a memory buffer and mix old data with new data to prevent forgetting.

Continual Embedded Learning. While many continual learning methods have been developed to overcome forgetting, some make strong assumptions which make them ill-suited for embedded applications. For example, many algorithms train using large batches consisting of thousands of training examples and if trained instance-by-instance, their performance degrades (Hayes et al., 2020). Batch learning is undesirable for embedded applications since it requires time to queue up batches and more time to train. For example, batch learning methods in Hayes & Kanan (2020) required over 60 hours to train, while the proposed online learning method required only 30 minutes. Further, while looping over batches to train, the model cannot make inferences, which can be prohibitive in real-time scenarios. Moreover, some algorithms require task labels to be provided during inference and if this assumption is violated, their performance also degrades (Hayes & Kanan, 2020; Hayes et al., 2020). Also relevant to this work are methods designed specifically for few-shot continual learning (Ayub & Wagner, 2020; 2021; Tao et al., 2020b;a). These methods consider the ability of algorithms to generalize from very few instances, which is one of the necessary capabilities for continual learning on embedded devices. In this work, we focus on studying online continual learning algorithms that learn instance-by-instance in a single pass through the dataset, with memory and compute constraints, and without the need for task labels during inference. We believe this setup is representative of how agents would be trained and evaluated on embedded devices.

There has been a limited amount of research studying continual learning for embedded applications (Pellegrini et al., 2020; 2021; Demosthenous & Vassiliades, 2021; Li et al., 2019). Most focus on supervised classification using small batches on the CORe50 dataset (Lomonaco & Maltoni, 2017) with a MobileNet-v1 backbone (Pellegrini et al., 2020; 2021; Demosthenous & Vassiliades, 2021), while one focuses on continual object detection (Li et al., 2019). Small batches do not meet real-world on-device continual learning needs, where a user wants to minimize the number of samples. To address this, we study seven online continual learning algorithms that learn instance-by-instance. Each method is combined with five different backbone CNNs. We evaluate the performance of these methods on three unique datasets that test the robustness of algorithms to scale (i.e., resolution, number of classes, and number of training examples), imbalanced/long-tailed data streams, and temporally correlated video streams. Existing work has separately evaluated the robustness of continual learners to scale (Hou et al., 2019; Rebuffi et al., 2017; Castro et al., 2018; Wu et al., 2019), video streams (Hayes et al., 2020; Hayes & Kanan, 2020; Pellegrini et al., 2020; 2021), imbalanced data streams (Aggarwal et al., 2021; Belouadah et al., 2020; Hu et al., 2020), and low-shot learning (Ayub & Wagner, 2020; 2021; Tao et al., 2020b;a). Conversely, we provide the first comprehensive study that directly compares the robustness of online learners to each of these scenarios when paired with CNNs specifically designed for on-device learning. We compare methods in terms of classification efficacy, memory, and compute.

Efficient On-Device Learning. Several convolutional neural network (CNN) architectures have been designed to address the need for on-device learning. These architectures are designed to achieve high accuracy while also considering computational efficiency. Methods that have been proposed for improving efficiency and reducing memory (e.g., network parameters) include 1×1 convolutions (SqueezeNet) (Iandola et al., 2016), group convolutions (ShuffleNet, CondenseNet) (Zhang et al., 2018; Huang et al., 2018), depth-wise separable convolutions (MobileNet-v1, MobileNet-v2) (Howard et al., 2017; Sandler et al., 2018), and point-wise convolutions (ShiftNet) (Wu et al., 2018). Memory can also be reduced using pruning (Alvarez & Salzmann, 2016; Han et al., 2015; Hu et al., 2016; LeCun et al., 1989; Li et al., 2016; Louizos et al., 2017; Srinivas & Babu, 2015), quantization (Courbariaux et al., 2016; Jacob et al., 2018; Kim & Smaragdis, 2016), model compression (Jacob et al., 2018; Krishnamoorthi, 2018; Wu et al., 2016; Soudry et al., 2014; Zhou et al., 2017a; 2016; Rastegari et al., 2016), or network distillation (Bucilua et al., 2006; Hinton et al., 2015). Beyond this, reinforcement learning and neural architecture search have been used to find architectures that balance accuracy with efficiency (Zoph & Le, 2016; Zoph et al., 2018; Baker et al., 2016; Liu et al., 2018; Pham et al., 2018; Tan et al., 2019; Cai et al., 2018; Yang et al., 2018). In this paper, we use two common architectures designed to fit on mobile devices that achieve strong performance on supervised ImageNet classification: MobileNet-v3 (Howard et al., 2019) and EfficientNet (Tan & Le, 2019), which we discuss more in Sec. 4.1.

4 Experimental Setup

We incrementally train a neural network $\hat{y}_t = F\left(G\left(\mathbf{X}_t\right)\right)$ using supervised online continual learning, where $\mathbf{X}_t$ is an image at time t and $\hat{y}_t$ is the predicted label. Following traditional transfer learning setups, $G\left(\cdot\right)$ is a backbone CNN pre-trained using supervised learning on the ImageNet-1k dataset (Russakovsky et al., 2015) and $F(\cdot)$ is a classifier. We assume the output of $G(\cdot)$ is a vector, which can be obtained using global average pooling or other pooling mechanisms. In our setup, we freeze $G(\cdot)$ after pre-training and train $F(\cdot)$ using online updates. That is, we focus on updating the classifier in an online fashion (i.e., one example at a time) using fixed feature representations. While updating features could improve performance further, this requires additional compute time, which is undesirable in embedded settings where training and inference must happen with limited compute capacity. Using this setup, we study five backbone architectures for $G(\cdot)$ and seven online continual learning methods to update $F(\cdot)$, which we describe next.

4.1 Network Architectures

We study five backbone CNN architectures, which were chosen due to their small size, efficiency, and competitive performance when trained using supervised learning on ImageNet-1k.

MobileNet-v3 (Howard et al., 2019) builds on two previous MobileNet architectures designed for mobile and embedded applications (Howard et al., 2017; Sandler et al., 2018). Specifically, it combines depth-wise separable convolutions, inverted residual connections, squeeze and excitation attention modules (Iandola et al., 2016), neural architecture search (NAS), and more efficient nonlinearities. There are two versions of the CNN: *MobileNet-v3 (Small)*, which uses fewer resources and *MobileNet-v3 (Large)*, which uses more resources, but achieves higher classification accuracies. We study both versions of the architecture. The small architecture contains 927,000 parameters and achieves a top-1 accuracy of 67.7% on ImageNet-1k. The large architecture contains 2.91 million parameters and achieves a top-1 accuracy of 74.0% on ImageNet-1k.

EfficientNet (Tan & Le, 2019) uses fixed scaling coefficients to scale the width, depth, and resolution of a network with the amount of computational resources available. Similar to MobileNets, EfficientNets combine inverted residual connections and squeeze and excitation modules with NAS to find an architecture that balances accuracy with efficiency. We use the two smallest EfficientNets: *EfficientNet-B0* (4 million parameters) and *EfficientNet-B1* (6.5 million parameters). EfficientNet-B0 achieves a top-1 accuracy of 77.7% on ImageNet-1k, while EfficientNet-B1 achieves a top-1 accuracy of 78.6%.

ResNet-18 (He et al., 2016) is the smallest ResNet architecture commonly used in practice. This architecture was chosen since it is a common architecture used in continual learning literature (Rebuffi et al., 2017; Castro et al., 2018; Hayes et al., 2020; Douillard et al., 2020; Wu et al., 2019). It contains 11 million parameters and achieves a top-1 accuracy of 69.8% on ImageNet-1k.

For all architectures, we global average pool features prior to the classifier. This yields features of the following dimensions: MobileNet-v3 (Small) 576-d, MobileNet-v3 (Large) 960-d, EfficientNet-B0 1280-d, EfficientNet-B1 1280-d, and ResNet-18 512-d.

4.2 Online Continual Learning Models

We study seven online continual learning methods for updating the classifier $F(\cdot)$ using pre-trained universal image features from $G(\cdot)$. These methods were chosen due to their ability to learn one sample at a time in a single pass over a dataset without task labels (i.e., online continual learning) with low memory and compute requirements.

Fine-Tune incrementally fine-tunes a fully-connected output layer one sample at a time using stochastic gradient descent and a cross-entropy loss. This method does not have any mechanisms to prevent catastrophic forgetting.

Nearest Class Mean (NCM) maintains one running mean vector per class (i.e., $\mathbf{w}_k$ is the mean for the k-th class), each with an associated counter denoting the number of samples represented in each mean (c_k). Given a new data vector $\mathbf{x}_t$ with associated label y_t at time t, the class mean and associated counter are updated as:

$$\mathbf{w}_{y_t} \leftarrow \frac{c_{y_t}\mathbf{w}_{y_t} + \mathbf{x}_t}{c_{y_t} + 1} \ , \qquad c_{y_t} \leftarrow c_{y_t} + 1 \ . \tag{1}$$

During inference, it assigns the label of the nearest class mean to a new example. This is a common baseline in continual learning (Rebuffi et al., 2017). Following past work we use Euclidean distance for the metric.

Streaming One-vs-Rest (SOvR) compares how close a new example is to a class mean vector while also considering its distance to data from other classes, which is reminiscent of support vector machines. Specifically, it maintains one running mean vector and associated count per class, which are updated using Eq. 1. During inference, the method computes one vector per class ($\tilde{\mathbf{w}}_k$) that is representative of the mean of data not belonging to that class, i.e.,

$$\tilde{\mathbf{w}}_k \leftarrow \frac{1}{N} \sum_{i \neq k} c_i \mathbf{w}_i \quad , \tag{2}$$

where $N = \sum_i c_i$ is the sum of all class counts. To assign a label to a new example, the SOvR method first computes the distance of the new sample to each vector $\mathbf{w}_k$ as d_k and to each vector $\tilde{\mathbf{w}}_k$ as $\tilde{d}_k$ by computing the dot product between the new sample with each of the vectors. The score for class k is then computed as $s_k = \frac{d_k}{d_k + \tilde{d}_k}$ and a label is assigned by taking the argmax over all s_k.

Streaming Linear Discriminant Analysis (SLDA) maintains one running mean vector per class with an associated counter, which are updated using Eq. 1, and one shared running covariance matrix among classes. We use the implementation from Hayes & Kanan (2020) to update the covariance matrix and compute predictions. This implementation was shown to work well on the large-scale ImageNet-1k dataset. Intuitively, SLDA makes predictions by assigning a new example the label of the closest Gaussian in feature space defined using the running class means and shared covariance matrix. NCM is a special case of SLDA where the covariance matrix is equal to the identity matrix.

Streaming Gaussian Naive Bayes is related to Streaming Quadratic Discriminant Analysis (SQDA). While SQDA stores one running covariance matrix per class, Gaussian Naive Bayes instead stores one variance vector per class (i.e., diagonal covariance matrices that assume independent features). It also stores one running mean vector per class with an associated counter. The advantage of Gaussian Naive Bayes to SQDA is that storing diagonal covariance matrices requires significantly less memory than storing full covariance matrices. Compared to SLDA, Gaussian Naive Bayes is able to more accurately represent the variance for each class instead of using a shared covariance matrix. However, unlike SLDA and SQDA, Gaussian Naive Bayes assumes feature independence with its variance vectors, which is not always guaranteed in practice. We use Eq. 1 to update the running mean vectors and Welford's sample variance algorithm (Welford, 1962) to update the running variance vectors in a numerically stable way. We perform inference using the same procedure as SLDA but with the per class variance vectors (i.e., diagonal covariance matrices) instead of a shared covariance matrix.

Online Perceptron maintains one weight vector for each class. The first time a sample from a new class is seen, the weight vector for the class is set to the sample. After that, each time the model misclassifies a new sample $\mathbf{x}_t$ with label y_t, the associated class weight vector is updated as:

$$\mathbf{w}_{y_t} \leftarrow \mathbf{w}_{y_t} + \mathbf{x}_t \quad , \tag{3}$$

and the weight vector of the incorrect class with the highest score ($\mathbf{w}_i$) is updated as

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - \mathbf{x}_t \quad . \tag{4}$$

During inference, a score for class k is computed by taking the dot product between $\mathbf{w}_k$ and an input vector $\mathbf{x}_t$. A label is then assigned by taking the argmax over the computed scores for all k classes.

Replay mitigates catastrophic forgetting by storing a subset of previous examples in a memory replay buffer and has grown in popularity due to its strong performance on large-scale continual learning benchmarks (Rebuffi et al., 2017; Castro et al., 2018; Hayes et al., 2020; Douillard et al., 2020). Specifically, replay maintains a memory buffer of size B, which equally distributes examples among classes seen so far. When the buffer is full, it randomly replaces an example from the most represented class with a new example. During training, it randomly selects P examples from the replay buffer, combines them with the new example, and makes a single update using stochastic gradient descent with a cross-entropy loss. Consistent with the other algorithms, we use replay to train a single fully-connected output layer. While effective, replay can be memory intensive due to the storage of its memory buffer.

4.3 Datasets

We use three high-resolution image classification datasets to evaluate the online continual learners.

OpenLORIS is a dataset containing videos of different household objects taken with a camera attached to a robot (She et al., 2020). Specifically, it contains 40 unique object classes each with videos of 1 to 9 different object instances. There are a total of 121 object instances. Each object instance was recorded under four varying domain factors: amount of clutter, amount/source of illumination, amount of occlusion, and pixel size of the object in frame. For each domain, the object instances were recorded under 9 different sessions of varying difficulty, yielding 36 unique videos for each object instance. The dataset contains over 440,000 training images and over 53,000 test images. This dataset is the most realistic for our setup because it requires agents to learn from non-stationary (temporally correlated) video streams and adapt to changing domains. Example images from the OpenLORIS dataset are in Fig. 2.

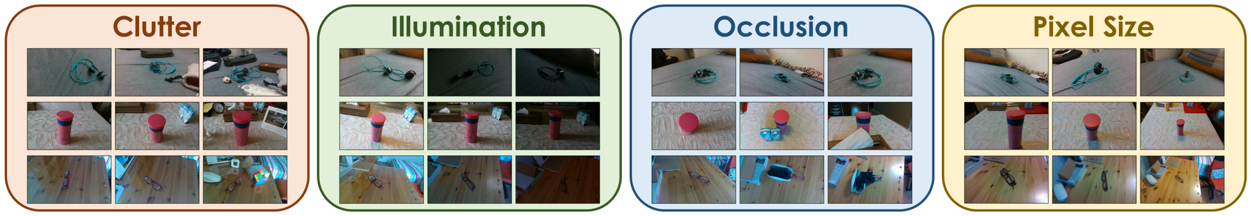


Figure 2: Example images from the OpenLORIS video dataset (She et al., 2020). We demonstrate the varied difficulty across each of the four domain factors for three unique object instances (headphones, bottle, and glasses).

Places-365 is a scene classification dataset containing over 1.8 million static images from 365 unique classes (Zhou et al., 2017b). We use the small image version of the dataset where images have been resized to 256×256 pixels. We evaluate on the full validation set because the test set is behind a server. The train set contains between 3,068 and 5,000 images per class, while the validation set contains 100 images per class. We use Places-365 to evaluate how well models scale in performing continual learning on over one million high-resolution images from hundreds of classes.

Places-Long-Tail (Places-LT) is a long-tailed version of the static Places-365 dataset which is used to test how well algorithms work in imbalanced regimes (Liu et al., 2019). It contains 62,500 total training images with 5 to 4,980 images per class and 365 total classes. Similar to Places-365, we use the validation set for testing. We use Places-LT to test the robustness of online continual learners to heavily imbalanced datasets, which are more representative of the distribution of objects in the natural world.

4.4 Data Orderings

Since the order in which data is presented to an online continual learner can potentially impact its performance, we study several different orderings of each dataset. Since OpenLORIS contains videos, we study two variants of the *instance* data ordering from Hayes et al. (2019). The instance ordering presents videos of shuffled object instances to the learner one at a time. For example, the learner could learn a video of mug #1, then a video of hat #3, then a video of mug #2, and so on. The learner is then evaluated on the full test set. The instance ordering is the most realistic ordering used in this paper since an agent could be expected to learn from temporally correlated video streams with the potential of revisiting previous classes. While the standard instance ordering presents *all* videos from the dataset to the learner, we also study a smaller variant of the instance ordering that we term "low-shot instance." In the *low-shot instance* ordering, the learner is presented with a single video of a single object instance from each category in the dataset. After each instance is learned, the learner is evaluated on all test data belonging to the classes seen so far. Since the learner is only trained on a single video from each category and evaluated on the entire test set for the category, it must be capable of generalizing to out-of-domain inputs to perform well. This is because the test set contains videos of the same object instance under different domains (e.g., clutter, occlusion, illumination, or size of object in frame), as well as videos of other object instances from the same class under different domains. Specifically, the low-shot instance ordering of OpenLORIS requires models to learn from naturally temporally correlated data streams, generalize to different object instances from the same classes, generalize to unseen domains, and learn from very little labeled data. This setting mimics a practical setting where a user might have a particular pet cat in their home that they would like a classifier to identify. If the user only provides the classifier with a few images of the cat on a couch, it would be ideal if the classifier could still identify the same pet cat in other domains (i.e., not sitting on the couch). Moreover, in addition to using the low-shot instance ordering of OpenLORIS to understand how well each continual learner can generalize from very little labeled data, we also perform experiments on the F-SIOL-310 (Few-Shot Incremental Object Learning) dataset (Ayub & Wagner, 2021) designed specifically for few-shot continual learning in supplemental materials.

Since Places-365 and Places-LT contain static images, we study two data orderings: *iid* where all of the images are randomly shuffled and *class-iid* where all of the images are organized by class, but shuffled within each class. In class-iid, classes are also shuffled. The iid ordering should be easiest since machine learning models typically assume data is shuffled and models are exposed to the same classes at multiple points during training. In contrast, the non-stationarity of the class-iid ordering tests the ability of models to overcome catastrophic forgetting and is a common benchmark in continual learning (Rebuffi et al., 2017; Castro et al., 2018; Hou et al., 2019; Wu et al., 2019). Models that suffer from catastrophic forgetting typically struggle with the class-iid ordering, but perform well on the iid ordering.

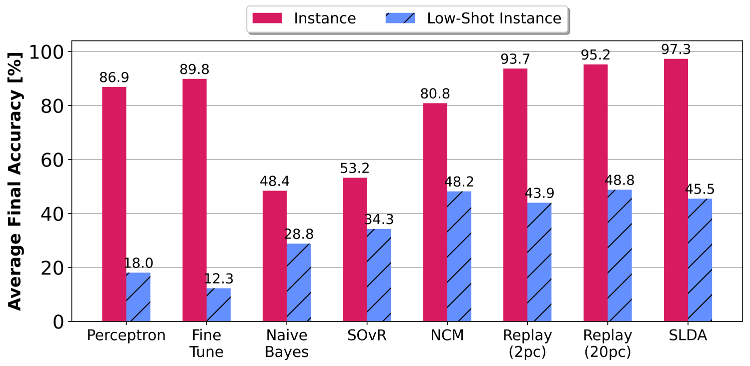


Figure 3: Final accuracy (%) summary statistics aggregated across CNN architectures to compare online continual learning methods when trained on the OpenLORIS dataset using the instance and low-shot instance data orderings.

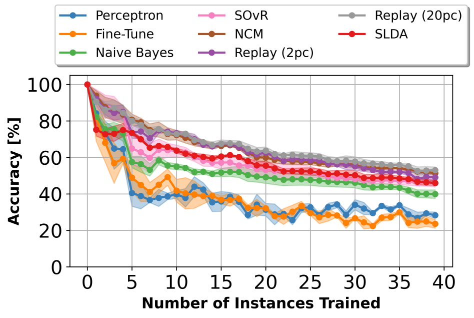


Figure 4: Learning curves of each online continual learner with the EfficientNet-B1 CNN when evaluated using the low-shot instance ordering of the OpenLORIS dataset. Each plot is the average over 3 runs with different instance permutations with standard error denoted by the shaded regions.

4.5 Performance Metrics

We evaluate online continual learner performance on three axes: classification efficacy, memory, and compute. For classification efficacy, we use a learner's final accuracy. For scoring methods, we adopt a modified variant of the NetScore from Wong (2019), which provides a single score that combines accuracy, total number of parameters, and amount of compute. It is useful to compute a single metric that considers all three factors in order to better analyze the applicability of each continual learner for embedded applications. For example, consider one learner that achieves strong classification efficacy using a lot of memory and compute and consider another learner that achieves similar classification efficacy using much less memory and compute. For embedded applications, the learner that uses less memory and compute would be desirable.

The NetScore metric takes inspiration from the decibel scale used in signal processing, which compares the ratio of one value as compared to another on a logarithmic scale. Specifically, the NetScore for online learners $\mathcal{M}$ is:

$$\Omega\left(\mathcal{M}\right) = s \log\left(\frac{a\left(\mathcal{M}\right)^{\alpha}}{p\left(\mathcal{M}\right)^{\beta} c\left(\mathcal{M}\right)^{\gamma}}\right) \quad , \tag{5}$$

where $a\left(\mathcal{M}\right)$ is a method's final accuracy, $p\left(\mathcal{M}\right)$ is the total number of parameters required to store both the CNN and the online continual learner, $c\left(\mathcal{M}\right)$ is the number of seconds required to run the experiment, and α, β, and γ are hyper-parameters that control the influence that accuracy, memory, and compute have on the NetScore Ω, respectively. Following Wong (2019), we set $\alpha = 2$ to emphasize the importance of high classification accuracy, and $s = 20$. We set $\beta = \gamma = 0.25$ due to the large scale of $p\left(\mathcal{M}\right)$ and $c\left(\mathcal{M}\right)$ to ensure mostly non-negative values of Ω. We include NetScore values using the original parameters from Wong (2019) in supplemental materials.

5 Results

We provide all implementation details in supplemental materials. For the OpenLORIS and Places-LT datasets, we run each online continual learner with three permutations of each data ordering and report the average performance over runs. Due to the large size of the Places-365 dataset, we run each online continual learning method with each ordering once. Note that the SOvR and NCM classifiers are agnostic to these data order permutations since their running mean estimates are unaffected by data order. We study two replay buffer sizes: one that stores 20 examples per class (20pc) and one that stores 2 examples per class (2pc) (see supplemental materials for more details).

5.1 Results on OpenLORIS

We first studied the performance of online continual learners on the realistic OpenLORIS video dataset under two data orderings: instance and low-shot instance. Recall in the instance ordering that each learner is trained on *all* object instance videos, while the low-shot instance ordering only provides the learners with a single object instance

Table 1: Ω NetScore values for each online continual learner using the *low-shot instance* ordering of the OpenLORIS dataset. We also report the mean accuracy (%) aggregated across CNN architectures. We format the **first**, **second** and third best overall results.

Method	MNet-S	MNet-L	ENet-B0	ENet-B1	RN-18	Mean
Perceptron	-12.3	3.0	20.6	18.6	-31.8	-0.4
Fine-Tune	-45.5	-33.4	15.2	11.4	-71.8	-24.8
Naive Bayes	21.1	33.4	36.9	31.6	-87.4	7.1
SOvR	25.1	27.6	39.1	37.2	6.5	27.1
NCM	48.0	44.9	46.3	42.5	37.4	**43.8**
Replay (2pc)	44.2	40.9	44.4	40.6	28.7	39.8
Replay (20pc)	46.7	44.2	46.1	43.0	35.8	**43.2**
SLDA	46.8	41.8	40.9	37.0	35.4	40.4

video from each of the 40 classes. To directly compare how these orderings affect model performance, we aggregated the final top-1 accuracy scores of each online continual learner across all CNN architectures in Fig. 3. As expected, all models perform worse when trained using the low-shot instance ordering. However, the performance differences for some models are much larger than others. For example, the perceptron and fine-tune models achieve strong performance when trained on all object instance videos, but perform poorly when only trained on a single object instance from each class. This indicates that these models exhibit poor generalization to out-of-domain inputs. While the Naive Bayes and SOvR methods outperform the perceptron and fine-tune methods in the low-shot instance setting, their performance in the full instance setting is the poorest among all methods. Overall, the replay (20pc) and SLDA methods strike the best compromise between strong performance when trained using all object instances and ability to generalize well when trained with very few instances. While replay (2pc) performs worse than replay (20pc), it still achieves competitive performance with replay (20pc) and SLDA, while requiring less memory.

To study performance differences among learners in the low-shot instance setting further, we show learning curves for each method with the EfficientNet-B1 backbone in Fig. 4. Learning curves with additional CNN backbones are in supplemental materials. Specifically, we show the top-1 accuracy of each method on classes seen so far after viewing each object instance video. Overall, the replay (20pc), replay (2pc), and NCM methods perform consistently the best across all training increments. While results vary slightly across CNN architectures, the replay (20pc) and NCM methods are consistently the top performers, indicating that these two methods generalize the best to out-of-domain inputs (i.e., object instances filmed under alternative clutter, occlusion, illumination, or pixel size settings).

5.1.1 NetScore Performance

Online continual learning on embedded devices poses unique challenges to learners; they must strike a balance between achieving strong performance, while also operating under memory and compute constraints. To better understand the practical usage of each method, we report the Ω NetScore values of continual learners with each CNN in Table 1. All methods were evaluated on the same hardware for consistency. Overall, the NCM method performed best, followed by replay (20pc) and SLDA. Since the NCM method only requires the storage and updates of class mean vectors, it is both computationally and memory efficient. While replay (20pc) and SLDA achieve strong performance in Fig. 3, their storage requirements and compute times are longer than NCM, meaning they have lower NetScores. Fine-tune performs the worst overall, followed by the perceptron and Naive Bayes. Interestingly, Naive Bayes achieves moderate Ω scores with the EfficientNet-B0, MobileNet-v3 (Large), and EfficientNet-B1 backbones and a poor score with the ResNet-18 backbone. This could be because the features learned with ResNet-18 are not independent, which is an assumption of the Naive Bayes method. Unsurprisingly, all methods achieve the worst Ω scores with the ResNet-18 architecture, which is because it is the largest network and requires the most memory and compute.

5.1.2 Backbone CNN Comparisons

Moreover, we were interested in seeing how performance varied for each online continual learning method across CNN architectures. To study this, we show the final top-1 accuracy scores achieved by each method when trained using the full instance ordering in Table 2. For nearly all online continual learners (with the exception of NCM), performance using the MobileNet-v3 (Small) and ResNet-18 backbones is the worst, while performance using the EfficientNet-B0 and EfficientNet-B1 backbones is the best. This is interesting as many existing continual learning methods have focused on pairing new algorithmic components for overcoming catastrophic forgetting with the ResNet-18 architecture (Rebuffi et al., 2017; Hou et al., 2019; Wu et al., 2019; Hayes et al., 2020). Since performance is better

Table 2: Final accuracy (%) values for each online continual learner using the *instance* ordering of the OpenLORIS dataset. We also report the mean accuracy (%) aggregated across CNN architectures. We format the **first**, **second** and third best overall results.

METHOD	MNET-S	MNET-L	ENET-B0	ENET-B1	RN-18	MEAN
Perceptron	79.3	88.0	93.5	94.2	79.6	86.9
Fine-Tune	83.5	91.5	95.8	96.3	82.1	89.8
Naive Bayes	31.1	52.6	78.0	78.8	1.5	48.4
SOvR	37.4	47.7	73.9	72.4	34.6	53.2
NCM	72.9	78.9	85.9	86.7	79.7	80.8
Replay (2pc)	89.3	94.2	97.0	97.4	90.7	93.7
Replay (20pc)	92.1	95.6	97.7	97.8	92.9	**95.2**
SLDA	95.6	98.2	98.8	98.8	95.0	**97.3**

using EfficientNets and they require less memory and compute than ResNet-18, we urge future researchers to consider alternative architectures for studying continual learning.

5.2 OVERALL RESULTS ON PLACES-365 AND PLACES-LT

Next, we looked at which continual learning method was the most effective, regardless of the CNN architecture, on both variants of the Places dataset. Average top-1 accuracies on Places-365 and Places-LT for all continual leaning methods averaged across CNN architectures are given in Table 3, where the overall accuracy is computed as the harmonic mean of the class-iid and iid runs to emphasize that it is important to do well on both extremes. The raw top-1 accuracy values with each CNN backbone are in supplemental materials. SLDA performed best overall, followed by replay (20pc) and NCM. As expected, in the iid setting, fine-tune performed well and in the class-iid setting it suffered from severe catastrophic forgetting. Similarly, the perceptron performed well on the iid ordering, but struggled with the class-iid ordering. This is likely because when the perceptron misclassifies an example, it updates the weight for the correct class and the weight for the highest predicted incorrect class. Since classes are not revisited in the class-iid setting, it is likely that the perceptron misclassifies new examples as previous classes and updates the previous class weight vectors. This leads to perturbations in the previous weights, causing catastrophic forgetting. While Naive Bayes and SOvR were unaffected by data ordering, their performance was much worse than the top-performing methods. For Naive Bayes, this is likely because its feature independence assumptions are violated.

Since Places-365 and Places-LT use the same test set, but different training sets, we can directly compare the performance of online continual learners on the two datasets to see how robust they are to dataset imbalance. Interestingly, the NCM method only exhibits a 3.1% loss in performance when trained on the long-tailed dataset, while the SLDA and replay (20pc) methods have larger gaps of 7.8% and 7.0%, respectively. This indicates that the NCM method is more robust to dataset imbalance than SLDA or replay (20pc). This is likely because SLDA assumes equal class covariances, which could be violated in the imbalanced regime. Similarly, performance of the replay method could potentially be improved by selectively replaying more examples from underrepresented classes. While most methods exhibit worse performance when trained on the long-tailed dataset, the perceptron and fine-tune methods exhibit minor performance gains, but still perform poorly overall. While we randomly shuffled classes for the Places-365 and Places-LT datasets, it could be an interesting future study to understand how the order of classes impacts performance on long-tailed datasets (e.g., ordering classes based on their number of training examples).

6 DISCUSSION AND CONCLUSION

Here we described a real-world problem where continual learning is needed: learning on embedded devices. From this need, we are able to specify the essential capabilities for these methods. In general, many existing continual learning frameworks do not meet the needs of embedded devices since they require processing data in large batches, require task labels to be provided during inference, or cannot learn from data streams presented in any order. To address the needs of continual learning for embedded devices, we compared seven online continual learning methods when paired with backbone CNN architectures designed for mobile and embedded applications to identify which methods fit our proposed criteria. We conducted experiments on three high-resolution image classification datasets to evaluate the robustness of the learners to scale (Places-365), imbalanced data streams (Places-LT), and realistic video data streams of specific object instances filmed under various domain settings (OpenLORIS). We compared learners across three axes: classification efficacy, memory, and compute. Overall, we found that the replay (20pc) and

Table 3: Final accuracy (%) summary statistics aggregated across CNN architectures to compare online continual learning methods. The iid and class-iid results are computed with the arithmetic mean across CNN architectures and the overall performance is computed as the harmonic mean (H-Mean) of these two numbers. An ideal method would achieve strong results regardless of ordering. We format the **<u>first</u>**, **second** and <u>third</u> best overall results.

	Places-365			Places-LT		
Method	iid	class-iid	H-Mean	iid	class-iid	H-Mean
Perceptron	32.2	0.9	1.8	18.0	4.1	6.7
Fine-Tune	44.0	2.9	5.4	20.4	5.0	8.1
Naive Bayes	12.7	12.7	12.7	9.6	9.6	9.6
SOvR	20.0	20.0	20.0	17.8	17.8	17.8
NCM	33.9	33.9	<u>33.9</u>	30.8	30.8	**30.8**
Replay (2pc)	43.4	20.7	28.1	29.7	20.6	24.3
Replay (20pc)	44.1	32.3	**37.3**	31.4	29.2	<u>30.3</u>
SLDA	39.3	39.3	**<u>39.3</u>**	31.5	31.5	**<u>31.5</u>**

SLDA models achieve strong performance on both orderings of the OpenLORIS dataset. However, when memory and compute time are factored in, the NCM method strikes the best trade-off between classification efficacy and efficiency on the OpenLORIS dataset. We also found that methods performed consistently the best when paired with an EfficientNet CNN. Finally, we found that the SLDA, replay (20pc), and NCM methods performed best on both variants and orderings of the Places dataset. We urge future embedded continual learning researchers to consider using the NCM, SLDA, and replay (20pc) models as baselines as they achieve strong classification efficacy while also minimizing memory and compute.

While we investigated the practicality of several online continual learners for embedded applications, there are several future research directions to explore. First, we focused on studying online learners that use fixed features from CNNs trained using supervised learning on ImageNet-1k. Recently, features learned using self-supervised learning have demonstrated strong performance on many downstream tasks (Chen et al., 2020a;b; Grill et al., 2020; Caron et al., 2020; Gallardo et al., 2021) and it would be interesting to apply them to embedded devices. The challenge is that most self-supervised techniques have been designed for large CNNs (e.g., ResNet-50) and perform poorly when applied directly to mobile architectures (Fang et al., 2021; Abbasi Koohpayegani et al., 2020). Moreover, we extracted pre-trained features from only the penultimate layer of each CNN. It could be interesting to explore the use of features from additional layers of the CNN as well. Moreover, the ImageNet dataset has its own limitations and might not yield the best features for every application, so it would be interesting to explore additional pre-training datasets.

Alternatively, methods that update feature representations could be considered in the future. There are two main challenges with continually updating representations for embedded applications: updating more features requires more memory and compute and updating feature representations can result in concept drift. Beyond this, improvements could be made to algorithms to better handle out-of-domain generalization and low-shot learning. While we found some methods performed better on the low-shot instance order than others, performance across all methods was worse than training on all object instances indicating the need for more robust low-shot learners. Moreover, we focused on supervised image classification, but additional capabilities such as object detection or segmentation could be studied for embedded devices in the future. For example, object detection requires both image classification and regression. The online classification methods explored in this paper could be combined with streaming regression models to perform lightweight, online object detection. Beyond this, we focused on CNN architectures designed for mobile and embedded applications; however, alternative methods to improve efficiency could be considered (e.g., pruning, quantization, etc.).

We believe that implementing supervised online continual learning algorithms on embedded devices requires the general criteria outlined in this paper. We established baselines for online continual learning methods to better understand which algorithms work best under a variety of settings that are useful for embedded applications (e.g., imbalanced data streams, large-scale data streams, video streams, and low-shot video streams). The insights learned from these experiments can provide future researchers with a starting point for specific online embedded applications such as AR/VR headsets, smartphones, robots, smart toys, and more. Performing embedded continual learning can potentially reduce latency, power consumption, privacy concerns, and the overall carbon footprint. Some challenges that will need to be considered include the efficient implementation of algorithms on specialized hardware and the background collection and processing of data that is provided to the continual learners.

ACKNOWLEDGMENTS

This work was supported in part by NSF awards #1909696, #2047556, and #DGE-2125362. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements of any sponsor. We thank James Arnold and Robik Shrestha for their comments and useful discussions.

REFERENCES

Soroush Abbasi Koohpayegani, Ajinkya Tejankar, and Hamed Pirsiavash. Compress: Self-supervised learning by compressing representations. *NeurIPS*, 33:12980–12992, 2020.

Umang Aggarwal, Adrian Popescu, Eden Belouadah, and Celine Hudelot. A comparative study of calibration methods for imbalanced class incremental learning. *Multimedia Tools and Applications*, pp. 1–20, 2021.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, pp. 139–154, 2018.

Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. *NeurIPS*, 29, 2016.

Ali Ayub and Alan R Wagner. Cognitively-inspired model for incremental learning using a few examples. In *CVPRW*, pp. 222–223, 2020.

Ali Ayub and Alan R Wagner. F-siol-310: A robotic dataset and benchmark for few-shot incremental object learning. In *ICRA*, pp. 13496–13502. IEEE, 2021.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In *ICCV*, pp. 583–592, 2019.

Eden Belouadah, Adrian Popescu, Umang Aggarwal, and Léo Saci. Active class incremental learning for imbalanced datasets. In *ECCVW*, pp. 146–162. Springer, 2020.

Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*, 2020.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *ECCV*, pp. 233–248, 2018.

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, pp. 532–547, 2018.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *ICLR*, 2019.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020a.

Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. In *NeurIPS*, 2020b.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Giorgos Demosthenous and Vassilis Vassiliades. Continual learning on the edge with tensorflow lite. *arXiv preprint arXiv:2105.01946*, 2021.

Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *CVPR*, pp. 5138–5146, 2019.

Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *ECCV*, 2020.

Zhiyuan Fang, Jianfeng Wang, Lijuan Wang, Lei Zhang, Yezhou Yang, and Zicheng Liu. {SEED}: Self-supervised distillation for visual representation. In *ICLR*, 2021.

Robert M French. Pseudo-recurrent connectionist networks: An approach to the 'sensitivity-stability' dilemma. *Connection Science*, 9(4):353–380, 1997.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

Jhair Gallardo, Tyler L Hayes, and Christopher Kanan. Self-supervised training enhances online continual learning. In *BMVC*, 2021.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS*, 2020.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *NeurIPS*, 28, 2015.

Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *CVPRW*, 2020.

Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In *ICRA*, pp. 9769–9776. IEEE, 2019.

Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *ECCV*, 2020.

Tyler L Hayes, Giri P Krishnan, Maxim Bazhenov, Hava T Siegelmann, Terrence J Sejnowski, and Christopher Kanan. Replay in deep learning: Current approaches and missing biological elements. *Neural Computation*, 33(11):2908–2950, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Lifelong learning via progressive distillation and retrospection. In *ECCV*, pp. 437–452, 2018.

Saihui Hou, Xinyu Pan, Zilei Wang, Chen Change Loy, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *CVPR*, 2019.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, pp. 1314–1324, 2019.

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 preprint arXiv:1704.04861*, 2017.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Xinting Hu, Yi Jiang, Kaihua Tang, Jingyuan Chen, Chunyan Miao, and Hanwang Zhang. Learning to segment the tail. In *CVPR*, pp. 14045–14054, 2020.

Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *CVPR*, pp. 2752–2761, 2018.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, pp. 2704–2713, 2018.

Ronald Kemker and Christopher Kanan. FearNet: Brain-inspired model for incremental learning. In *ICLR*, 2018.

Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

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*, 2017.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *NeurIPS*, 2, 1989.

Dawei Li, Serafettin Tasci, Shalini Ghosh, Jingwen Zhu, Junting Zhang, and Larry Heck. Rilod: Near real-time incremental learning for object detection at the edge. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pp. 113–126, 2019.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Zhizhong Li and Derek Hoiem. Learning without forgetting. In *ECCV*, pp. 614–629. Springer, 2016.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, pp. 19–34, 2018.

Ziwei Liu, Zhongqi Miao, Xiaohang Zhan, Jiayun Wang, Boqing Gong, and Stella X Yu. Large-scale long-tailed recognition in an open world. In *CVPR*, pp. 2537–2546, 2019.

Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *CoRL*, pp. 17–26, 2017.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, pp. 6467–6476, 2017.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.

Joseph O'Hagan and Julie R Williamson. Reality aware vr headsets. In *Proceedings of the 9TH ACM International Symposium on Pervasive Displays*, pp. 9–17, 2020.

Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jähnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *CVPR*, 2019.

German I Parisi, Jun Tani, Cornelius Weber, and Stefan Wermter. Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization. *Frontiers in Neurorobotics*, 12:78, 2018.

Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *IROS*, pp. 10203–10209. IEEE, 2020.

Lorenzo Pellegrini, Vincenzo Lomonaco, Gabriele Graffieti, and Davide Maltoni. Continual learning at the edge: Real-time training on smartphone devices. *arXiv preprint arXiv:2105.13127*, 2021.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, pp. 4095–4104. PMLR, 2018.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pp. 525–542. Springer, 2016.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *NeurIPS*, pp. 3738–3748, 2018.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv:1606.04671*, 2016.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pp. 4510–4520, 2018.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, pp. 4555–4564, 2018.

Qi She, Fan Feng, Xinyue Hao, Qihan Yang, Chuanlin Lan, Vincenzo Lomonaco, Xuesong Shi, Zhengwei Wang, Yao Guo, Yimin Zhang, et al. Openloris-object: A robotic vision dataset and benchmark for lifelong deep learning. In *ICRA*, pp. 4767–4773, 2020.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *NeurIPS*, 30, 2017.

Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. *NeurIPS*, 27, 2014.

Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pp. 6105–6114. PMLR, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, pp. 2820–2828, 2019.

Xiaoyu Tao, Xinyuan Chang, Xiaopeng Hong, Xing Wei, and Yihong Gong. Topology-preserving class-incremental learning. In *ECCV*, 2020a.

Xiaoyu Tao, Xiaopeng Hong, Xinyuan Chang, Songlin Dong, Xing Wei, and Yihong Gong. Few-shot class-incremental learning. In *CVPR*, pp. 12183–12192, 2020b.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *NeurIPS*, 29, 2016.

BP Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

Alexander Wong. Netscore: towards universal metrics for large-scale performance analysis of deep neural networks for practical on-device edge usage. In *International Conference on Image Analysis and Recognition*, pp. 15–26. Springer, 2019.

Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *CVPR*, pp. 9127–9135, 2018.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, pp. 4820–4828, 2016.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *CVPR*, pp. 374–382, 2019.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, pp. 285–300, 2018.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *ICLR*, 2018.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, pp. 3987–3995, 2017.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, pp. 6848–6856, 2018.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017a.

Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *TPAMI*, 2017b.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pp. 8697–8710, 2018.

A APPENDIX

A.1 IMPLEMENTATION DETAILS

We use the backbone CNN implementations and supervised ImageNet-1k pre-trained checkpoints for MobileNet-v3, EfficientNet, and ResNet-18 from torchvision. For the fine-tune and replay models, we select the best learning rate from $\{0.1, 0.01, 0.001, 0.0001\}$. For Places-365, the best learning rate is 0.0001. For Places-LT and OpenLORIS, the best learning rate is 0.001. For fine-tune and replay, we use a weight decay factor of 10^{-5}, a momentum of 0.9, and the stochastic gradient descent optimizer. For replay, we follow Gallardo et al. (2021) and randomly select 50 samples from the replay buffer to combine with the new sample to update the model. For replay, we study two buffer sizes: 1) storing 20 examples per class, which is common in continual learning literature (Rebuffi et al., 2017; Wu et al., 2019) and 2) storing 2 examples per class, which requires a similar amount of memory to the other online continual learners studied in this paper. For SLDA and Naive Bayes, we follow Hayes & Kanan (2020) and use a shrinkage value of 10^{-4}.

A.2 ADDITIONAL RESULTS

A.2.1 PLACES-365 AND PLACES-LT

Table 4: Final accuracy (%) results on the full Places-365 dataset with the iid and class-iid data orderings.

	IID					CLASS-IID				
METHOD	MNET-S	MNET-L	ENET-B0	ENET-B1	RN-18	MNET-S	MNET-L	ENET-B0	ENET-B1	RN-18
Perceptron	29.4	33.7	34.7	33.4	29.6	0.5	0.6	1.7	1.6	0.3
Fine-Tune	41.2	45.3	46.6	45.9	40.8	0.7	0.9	6.1	6.3	0.4
Naive Bayes	3.0	9.4	25.5	25.2	0.3	3.0	9.4	25.5	25.2	0.3
SOvR	9.6	16.9	29.1	28.3	16.0	9.6	16.9	29.1	28.3	16.0
NCM	29.4	34.0	36.8	36.4	32.8	29.4	34.0	36.8	36.4	32.8
Replay (2pc)	41.0	44.8	45.6	45.1	40.4	16.2	19.3	26.2	25.6	16.4
Replay (20pc)	41.3	45.7	46.3	45.7	41.5	29.5	32.6	35.8	35.0	28.8
SLDA	36.9	40.3	41.7	41.0	36.6	36.9	40.3	41.7	41.0	36.6

Table 5: Final accuracy (%) results on the long-tailed Places-LT dataset with the iid and class-iid data orderings. Each result is the average over 3 runs with different permutations of the data.

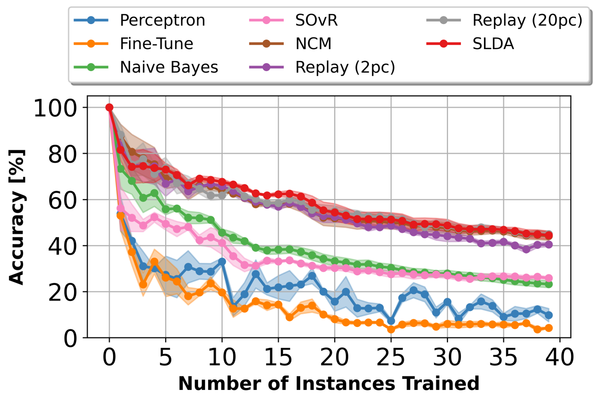
	IID					CLASS-IID				
METHOD	MNET-S	MNET-L	ENET-B0	ENET-B1	RN-18	MNET-S	MNET-L	ENET-B0	ENET-B1	RN-18
Perceptron	15.2	18.5	21.3	20.6	14.5	1.7	2.8	7.1	7.3	1.5
Fine-Tune	16.9	21.0	23.8	23.0	17.3	1.8	3.4	9.7	9.7	0.6
Naive Bayes	1.5	5.0	19.9	21.3	0.1	1.5	5.0	19.9	21.3	0.1
SOvR	8.9	14.9	26.2	24.5	14.6	8.9	14.9	26.2	24.5	14.6
NCM	26.5	31.0	33.6	32.9	30.0	26.5	31.0	33.6	32.9	30.0
Replay (2pc)	27.3	29.9	32.6	31.9	26.7	16.6	20.5	24.1	23.9	17.9
Replay (20pc)	29.4	32.2	34.0	33.1	28.5	26.6	29.5	31.9	31.3	26.8
SLDA	29.0	31.8	33.8	32.8	30.0	29.0	31.9	33.8	32.8	30.0

We include the final top-1 accuracies achieved by each online continual learning method with each CNN backbone on the Places-365 and Places-LT datasets in Table 4 and Table 5, respectively. These results are aggregated in Table 3.

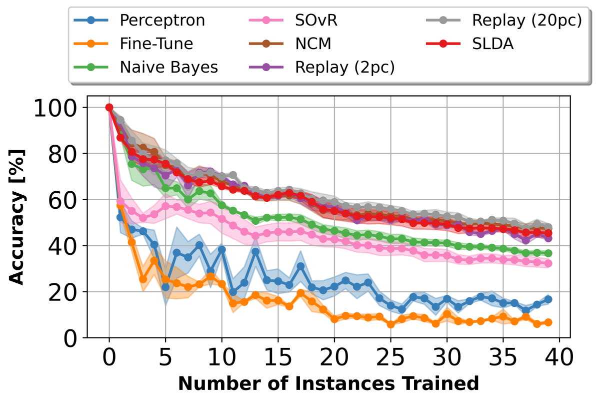
A.2.2 OPENLORIS

In Fig. 5, we show learning curves for each online continual learning method when trained with the low-shot instance ordering of the OpenLORIS dataset for various CNN backbones. Learning curves using the EfficientNet-B1 backbone are in Fig. 4. While results vary slightly across CNN architectures, the replay (20pc) and NCM methods are consistently the top performers. This indicates that NCM and replay (20pc) generalize the best to out-of-domain inputs.

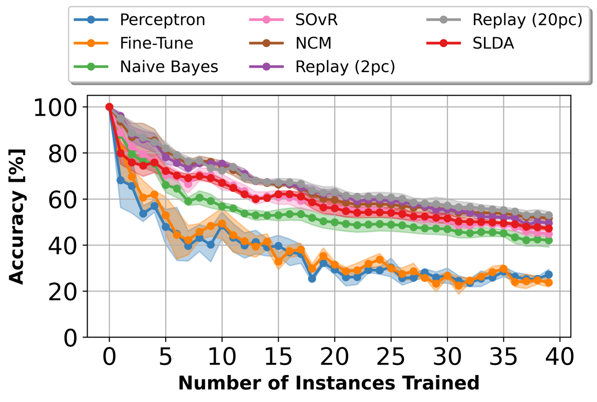
We show the final top-1 accuracy scores achieved by each method when trained using the low-shot instance ordering in Table 6. These scores coincide with the final accuracy values in Fig. 4 and Fig. 5.



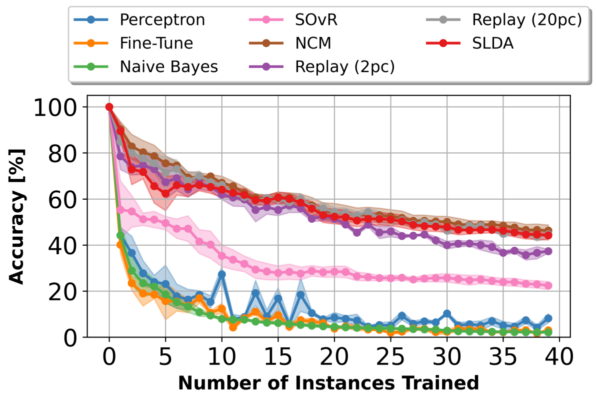
(a) MobileNet-v3 (Small)



(b) MobileNet-v3 (Large)



(c) EfficientNet-B0



(d) ResNet-18

Figure 5: Learning curves of each online continual learner with various backbone CNNs when evaluated using the low-shot instance ordering of the OpenLORIS dataset. Each plot is the average over 3 runs with different instance permutations. The standard error over runs is denoted by the shaded regions.

Table 6: Final accuracy (%) values for each online continual learner using the *low-shot instance* ordering of the OpenLORIS dataset. We also report the mean accuracy (%) aggregated across CNN architectures. We format the **<u>first</u>**, **second** and <u>third</u> best overall results.

METHOD	MNET-S	MNET-L	ENET-B0	ENET-B1	RN-18	MEAN
Perceptron	9.8	16.7	27.2	28.3	8.2	18.0
Fine-Tune	4.3	6.7	23.8	23.6	3.0	12.3
Naive Bayes	23.3	36.7	42.1	39.9	2.1	28.8
SOvR	25.9	32.3	44.9	45.9	22.4	34.3
NCM	44.2	47.4	51.6	51.4	46.3	**48.2**
Replay (2pc)	40.5	43.2	49.6	49.1	37.3	43.9
Replay (20pc)	45.0	48.1	53.0	53.0	44.7	**<u>48.8</u>**
SLDA	44.5	45.4	47.2	46.0	44.2	<u>45.5</u>

To compute the NetScore values in Table 1, we take into account a learner's classification efficacy (final accuracy), the total number of parameters required to store the backbone and continual learning model, and the compute time required to run the experiment in seconds. To examine how each of these factors contribute directly to a learner's NetScore, we show the raw efficacy, memory, and compute for each learner with each backbone in Tables 7-11.

For completeness, Table 12 contains NetScore values for each continual learner when computed using the original parameters suggested by Wong (2019), i.e., $\alpha = 2$, $\beta = \gamma = 0.5$. The NetScore values in Table 12 follow similar

Table 7: Classification efficacy (final accuracy), memory (number of parameters), compute (experiment run-time in seconds), and associated NetScore (Ω) values for each online continual learner using the *low-shot instance* ordering of the OpenLORIS dataset with the **MobileNet-v3 (Small)** backbone CNN. We format the **first**, **second** and third best overall results.

METHOD	EFFICACY	MEMORY	COMPUTE	Ω
Perceptron	9.8	950048	1041	-12.3
Fine-Tune	4.3	950048	1040	-45.5
Naive Bayes	23.2	996128	1260	21.1
SOvR	25.9	950048	1414	25.0
NCM	44.2	950048	1035	**48.0**
Replay (2pc)	40.5	996128	1053	44.2
Replay (20pc)	45.0	1410848	1052	46.7
SLDA	44.5	1281824	1040	**46.8**

Table 8: Classification efficacy (final accuracy), memory (number of parameters), compute (experiment run-time in seconds), and associated NetScore (Ω) values for each online continual learner using the *low-shot instance* ordering of the OpenLORIS dataset with the **MobileNet-v3 (Large)** backbone CNN. We format the **first**, **second** and third best overall results.

METHOD	EFFICACY	MEMORY	COMPUTE	Ω
Perceptron	16.7	3010352	1082	3.0
Fine-Tune	6.7	3010352	1084	-33.4
Naive Bayes	36.6	3087152	1329	33.4
SOvR	32.3	3010352	1583	27.6
NCM	47.4	3010352	1078	**44.8**
Replay (2pc)	43.2	3087152	1093	40.9
Replay (20pc)	48.1	3778352	1094	**44.2**
SLDA	45.4	3931952	1082	41.8

trends as the NetScore values using $\alpha = 2$, $\beta = \gamma = 0.25$ from Table 1. That is, the best performing model is NCM, followed by replay (20pc).

Next, we examine how robust the NetScore metric is to changes in dataset scale. We provide high-level proof-of-concept plots to examine how the NetScore metric scales when a dataset contains more samples and when a dataset contains more classes by interpolating our NetScore results from Table 1. Specifically, to understand how NetScore scales with the number of samples in a dataset, we use the original classification efficacy and memory (parameters) values from Table 1, but scale the compute (run-time) by multiples greater than one, e.g., scaling compute time by a factor of two would give us the compute time required to run the experiment on a dataset two times the size of the original dataset containing 440,919 train and 53,295 test samples. Similarly, to understand how NetScore scales with the number of classes, we use the original classification efficacy and compute (run-time) values from Table 1, but compute the number of parameters (memory) required by each model for datasets containing more classes. The resulting plots are in Fig. 6 and in Fig. 7 for datasets with more samples and more classes, respectively. As expected, when the number of samples or number of classes in a dataset increases, the online continual learner NetScore values monotonically decrease. However, some methods are more negatively impacted by the addition of more classes (e.g., replay) than others.

A.2.3 F-SIOL-310

Finally, we were interested to understand how each of the continual learning methods performed on a dataset designed specifically for few-shot continual learning. Specifically, we use the F-SIOL-310 (Few-Shot Incremental Object Learning) dataset (Ayub & Wagner, 2021), which consists of 620 total static images of 310 unique object instances from 22 unique classes. To perform online continual learning experiments using this dataset, we use the class-iid data ordering in both 5-Shot and 10-Shot learning scenarios, as suggested by Ayub & Wagner (2021). For the 5-Shot learning scenario, we randomly select five images from each class for training and the rest of the dataset is used for testing. For the 10-Shot learning scenario, we randomly select 10 images from each class for training and the rest of the dataset is used for testing. We run each experiment three times with different permutations of the classes and report the average

Table 9: Classification efficacy (final accuracy), memory (number of parameters), compute (experiment run-time in seconds), and associated NetScore (Ω) values for each online continual learner using the *low-shot instance* ordering of the OpenLORIS dataset with the **EfficientNet-B0** backbone CNN. We format the **first**, **second** and third best overall results.

METHOD	EFFICACY	MEMORY	COMPUTE	Ω
Perceptron	27.2	4058748	1204	20.6
Fine-Tune	23.8	4058748	1198	15.2
Naive Bayes	42.1	4161148	1475	36.9
SOvR	44.9	4058748	1635	39.1
NCM	51.6	4058748	1196	**46.2**
Replay (2pc)	49.6	4161148	1211	44.4
Replay (20pc)	53.0	5082748	1211	**46.1**
SLDA	47.2	5697148	1202	40.9

Table 10: Classification efficacy (final accuracy), memory (number of parameters), compute (experiment run-time in seconds), and associated NetScore (Ω) values for each online continual learner using the *low-shot instance* ordering of the OpenLORIS dataset with the **EfficientNet-B1** backbone CNN. We format the **first**, **second** and third best overall results.

METHOD	EFFICACY	MEMORY	COMPUTE	Ω
Perceptron	28.3	6564384	1511	18.6
Fine-Tune	23.6	6564384	1502	11.4
Naive Bayes	39.9	6666784	1719	31.6
SOvR	45.9	6564384	1777	37.2
NCM	51.4	6564384	1501	**42.5**
Replay (2pc)	49.1	6666784	1518	40.6
Replay (20pc)	53.0	7588384	1513	**43.0**
SLDA	46.0	8202784	1509	36.9

results over the three runs. We use the parameter settings from Sec. A.1 for all methods except replay. For replay, we use a maximum buffer size of 44 as suggested by Ayub & Wagner (2021).

In addition to the online continual learners studied in the rest of this paper, we also study an online variant of the **Centroid-Based Concept Learning** (CBCL) model designed specifically for few-shot continual learning (Ayub & Wagner, 2020). CBCL updates multiple centroids per class during training. Specifically, the first time a sample from a class is seen, it is stored as a centroid. After that, each time the model sees an example from a class, it computes the distance to the nearest centroid from that class. If the sample is within a threshold distance of the nearest centroid, it is merged with that centroid, otherwise it forms a new centroid. To perform inference, CBCL performs a weighted nearest neighbor computation where the class weight is assigned as the inverse of the number of examples that have been seen for a particular class. To make the algorithm amenable to online learning, we perform centroid updates on a sample-by-sample basis using Eq. 1, we assign class weights during testing according to the number of samples that have been seen for a class, and we merge the two closest class clusters into one cluster when the number of centroids needs to be reduced as:

$$\mathbf{w}_i = \frac{\mathbf{w}_i + \mathbf{w}_j}{c_i + c_j} \quad , \qquad c_i = c_i + c_j \quad , \tag{6}$$

where $\mathbf{w}_i$ and $\mathbf{w}_j$ are the two closest clusters with associated counts c_i and c_j. We use the parameters suggested by Ayub & Wagner (2021), i.e., a distance threshold of 17, a nearest neighbor k value of 1, and a maximum buffer size of 44 centroids.

The results for each continual learner with each backbone CNN are in Table 13. Overall, we found that the top-performing method across both the 5-Shot and 10-Shot learning scenarios was SLDA. The next top performers were NCM and CBCL. All three of these methods are distance-based classifiers, which have been shown to work well in low-shot learning settings (Snell et al., 2017; Vinyals et al., 2016). Although CBCL performed fairly well, it requires more memory and compute than NCM to update multiple centroids and has slightly worse performance, so NCM is still a strong method for this dataset. While the replay method was a top performer on other datasets, it struggles in the few-shot learning setting, which is likely due to not having enough replays of previous examples to mitigate

Table 11: Classification efficacy (final accuracy), memory (number of parameters), compute (experiment run-time in seconds), and associated NetScore (Ω) values for each online continual learner using the *low-shot instance* ordering of the OpenLORIS dataset with the **ResNet-18** backbone CNN. We format the **first**, **second** and third best overall results.

METHOD	EFFICACY	MEMORY	COMPUTE	Ω
Perceptron	8.2	11196992	1076	-31.8
Fine-Tune	3.0	11196992	1072	-71.8
Naive Bayes	2.1	11237952	1314	-87.4
SOvR	22.4	11196992	1567	6.5
NCM	46.3	11196992	1073	**37.4**
Replay (2pc)	37.3	11237952	1089	28.7
Replay (20pc)	44.7	11606592	1083	**35.7**
SLDA	44.2	11459136	1074	35.4

Table 12: Ω NetScore values for each online continual learner using the *low-shot instance* ordering of the OpenLORIS dataset. Here, we use $\alpha = 2$, $\beta = \gamma = 0.5$ to compute the NetScore as originally done in Wong (2019). We also report the mean accuracy (%) aggregated across CNN architectures. We format the **first**, **second** and third best overall results.

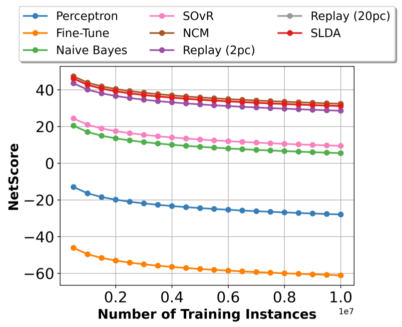
METHOD	MNET-S	MNET-L	ENET-B0	ENET-B1	RN-18	MEAN
Perceptron	-115.9	-106.5	-91.0	-96.5	-147.8	-111.5
Fine-Tune	-149.0	-142.9	-96.3	-103.7	-187.8	-135.9
Naive Bayes	-83.7	-77.3	-75.8	-84.2	-204.5	-105.1
SOvR	-80.0	-83.9	-74.0	-78.7	-111.4	-85.6
NCM	-55.5	-64.7	-65.3	-72.5	-78.7	**-67.3**
Replay (2pc)	-59.7	-68.8	-67.3	-74.6	-87.5	-71.6
Replay (20pc)	-58.9	-66.5	-66.6	-72.8	-80.5	**-69.1**
SLDA	-58.3	-69.1	-72.3	-79.3	-80.8	-72.0

forgetting. Based on these results, we recommend using distance-based classifiers for low-shot embedded continual learning applications. Specifically, it could be interesting to consider combinations of these methods in future work such as the combination of SLDA with multiple centroids as in CBCL.

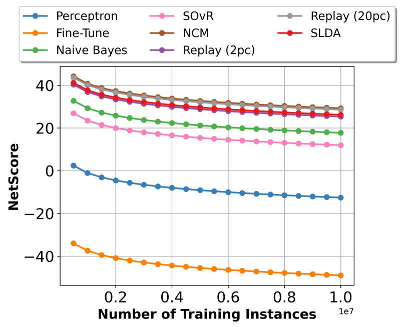
A.2.4 OVERALL RESULTS

To better understand the performance of individual methods across experiments, we create spider plots with the performance of each online continual learner with respect to NetScore, the ability to learn from temporally correlated videos, the ability to generalize from few labeled inputs, the ability to scale to large-scale data, and the ability to perform well on imbalanced data. To create these plots, we first average the performance of each continual learner across all five backbone CNNs. We then normalize NetScore values to fall in the range $[0, 1]$ such that the method with the best NetScore has a normalized score of 1 and the method with the worst NetScore has a normalized score of 0. For learner performance on videos, low-shot learning, ability to scale, and ability to learn from imbalanced data, we report the final scores achieved on the OpenLORIS (instance), OpenLORIS (low-shot instance), Places-365 (harmonic mean), and Places-LT (harmonic mean) datasets, respectively. The resulting spider plots are in Fig. 8, where the title of each plot contains the name of the online continual learner and its associated average performance across the five axes considered in the plots.

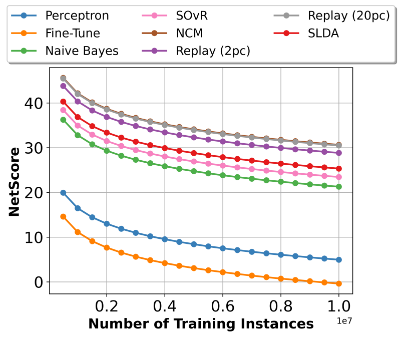
Overall, these plots indicate that SLDA and replay (20pc) performed the best across all five axes. NCM and replay (2pc) also had strong overall performance. While the fine-tune and perceptron methods performed the worst overall, they had strong performance on the video-based OpenLORIS dataset. In contrast, SOvR had better performance overall, but worse performance compared to fine-tune and perceptron on videos.



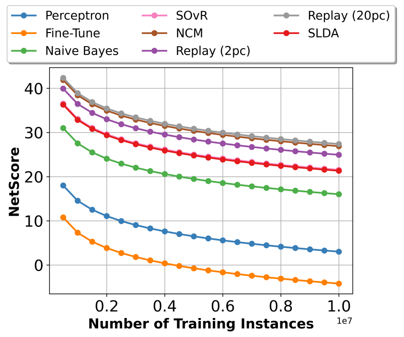
(a) MobileNet-v3 (Small)



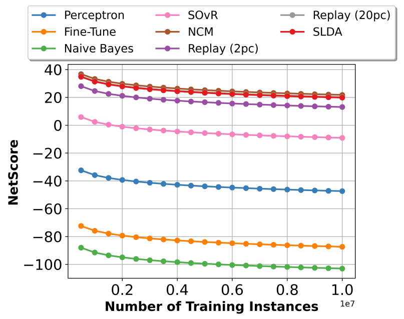
(b) MobileNet-v3 (Large)



(c) EfficientNet-B0



(d) EfficientNet-B1



(e) ResNet-18

Figure 6: NetScore values interpolated to account for additional samples in a dataset for each backbone CNN and continual learner.

Table 13: Final accuracy (%) results on the F-SIOL-310 dataset with the class-iid data ordering under the 5-shot and 10-shot learning scenarios. Each result is the average over 3 runs with different permutations of the data. We format the **first**, **second** and third best overall results.

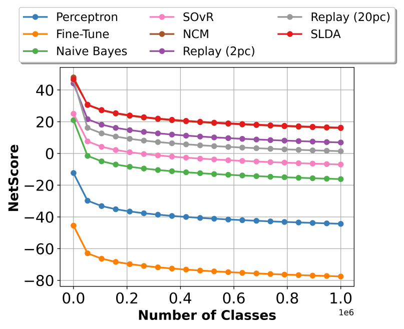
	5-Shot						10-Shot					
Method	MNet-S	MNet-L	ENet-B0	ENet-B1	RN-18	Mean	MNet-S	MNet-L	ENet-B0	ENet-B1	RN-18	Mean
Perceptron	19.4	21.6	35.0	39.5	5.0	24.1	12.7	18.7	34.4	51.0	5.3	24.4
Fine-Tune	18.0	22.9	41.5	48.4	9.0	28.0	12.4	19.8	36.2	43.8	8.2	24.1
Naive Bayes	26.7	50.7	79.2	82.9	3.9	48.7	34.8	56.4	79.2	85.2	1.8	51.5
SOvR	54.8	66.0	65.9	71.5	44.8	60.6	56.2	68.2	65.0	80.6	49.7	63.9
CBCL	82.9	85.8	83.9	80.2	81.5	82.9	90.9	91.8	91.5	90.4	89.9	90.9
NCM	82.9	85.9	85.3	86.2	85.0	**85.1**	90.9	91.8	91.9	92.0	90.8	**91.5**
Replay (2pc)	52.0	58.5	56.7	59.9	59.2	57.3	63.3	73.0	74.9	76.1	75.8	72.6
SLDA	88.5	90.7	90.3	89.0	85.4	**88.8**	93.9	96.6	96.6	94.7	91.5	**94.7**

A.3 Online Continual Learner Considerations

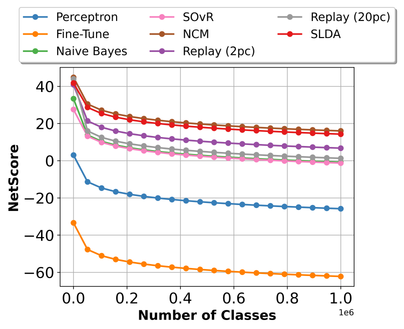
There are several strengths and weaknesses of each online continual learning method that must be considered when choosing methods for embedded applications. We highlight the strengths and weaknesses of each online continual learning method next.

Fine-Tune and **Perceptron** - Strengths: fast; only require memory to store the classifier; work well on iid data. Weaknesses: do not have mechanisms to mitigate catastrophic forgetting (i.e., do not work well across all data orderings); do not perform well on imbalanced data since they will likely overfit to overrepresented classes and underfit to underrepresented classes; fine-tune has many hyperparameters to tune (e.g., learning rate, weight decay, momentum, etc.).

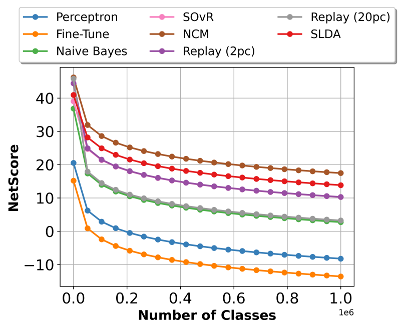
SOvR - Strengths: running means are order agnostic; no hyperparameters to tune. Weaknesses: does not work well in many of the experimental settings evaluated in this paper.



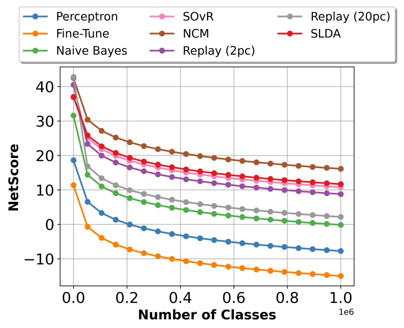
(a) MobileNet-v3 (Small)



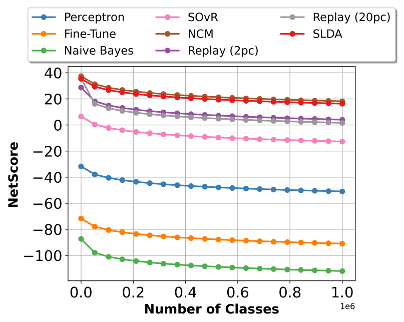
(b) MobileNet-v3 (Large)



(c) EfficientNet-B0



(d) EfficientNet-B1



(e) ResNet-18

Figure 7: NetScore values interpolated to account for additional classes in a dataset for each backbone CNN and continual learner.

Naive Bayes - Strengths: assumes different variances per class so it can model individual class distributions; fairly robust to data orderings since its running mean vectors are order agnostic and its class variance vectors will result in (at most) gradual forgetting; it is a generative classifier which could allow it to better model data from very few examples (i.e., low-shot settings). Weaknesses: assumes features are independent, which isn't always guaranteed; assumes data is Gaussian, which isn't always guaranteed.

NCM - Strengths: class mean vectors are order agnostic; no hyperparameters to tune; simple to implement. Weaknesses: cannot model outlier data.

SLDA - Strengths: fairly robust to data orderings since its running mean vectors are order agnostic and its covariance matrix will result in (at most) gradual forgetting; it is a generative classifier which could allow it to better model data from very few examples (i.e., low-shot settings); works well across a variety of experiments. Weaknesses: assumes data is Gaussian, which isn't always guaranteed; assumes classes have equal covariances, which isn't always guaranteed.

Replay - Strengths: fairly robust to data orderings since previous data is maintained in a memory buffer; works well across a variety of experiments. Weaknesses: requires storage of additional memory to work well; has many hyperparameters to tune (e.g., buffer size, replay selection method, learning rate, weight decay, momentum, etc.).

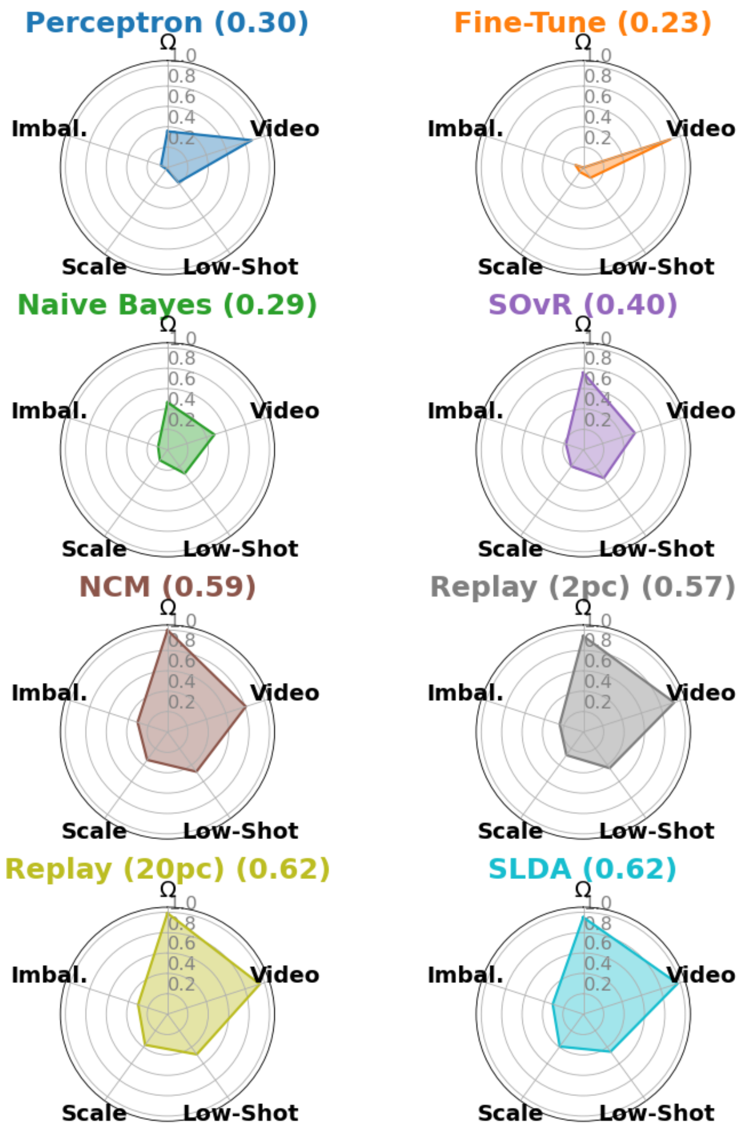


Figure 8: Spider plots comparing the normalized performance of each online continual learner with respect to NetScore (Ω), the ability to learn from temporally correlated videos (Video), the ability to generalize from few labeled inputs (Low-Shot), the ability to scale to large-scale data (Scale), and the ability to perform well on imbalanced data (Imbal). The average performance of each learner across all of these axes is at the top of each plot. Each metric has been averaged across all CNNs and normalized to fall in the range $[0, 1]$.