CarM: Hierarchical Episodic Memory for Continual Learning

Soobee Lee
UNIST
supersoob@unist.ac.kr

Minindu Weerakoon
UNIST
minindu1@unist.ac.kr

Jonghyun Choi
Yonsei University
jc@yonsei.ac.kr

Minjia Zhang
Microsoft
minjiaz@microsoft.com

Di Wang
Microsoft
wangdi@microsoft.com

Myeongjae Jeon
UNIST
mjjeon@unist.ac.kr

ABSTRACT

Continual Learning (CL) is an emerging machine learning paradigm in mobile or IoT devices that learns from a continuous stream of tasks. To avoid forgetting of knowledge of the previous tasks, episodic memory (EM) methods exploit a subset of the past samples while learning from new data. Despite the promising results, prior studies are mostly simulation-based and unfortunately do not promise to meet an insatiable demand for both EM capacity and system efficiency in practical system setups. We propose CarM, the first CL framework that meets the demand by a novel hierarchical EM management strategy. CarM has EM on high-speed RAMs for system efficiency and exploits the abundant storage to preserve past experiences and alleviate the forgetting by allowing CL to efficiently migrate samples between memory and storage. Extensive evaluations show that our method significantly outperforms popular CL methods while providing high training efficiency.

ACM Reference Format:
Soobee Lee, Minindu Weerakoon, Jonghyun Choi, Minjia Zhang, Di Wang, and Myeongjae Jeon. 2022. CarM: Hierarchical Episodic Memory for Continual Learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530587

1 INTRODUCTION

With the increasing demand for realistic on-device machine learning, recent years have witnessed a novel learning paradigm, namely continual learning (CL), for training neural networks (NN) with a stream of non-*i.i.d.* data. In such a paradigm, the neural network is incrementally learned with insertions of new tasks [17], *i.e.*, learning new knowledge from new tasks over time while retaining previously learned knowledge. The training process in CL resembles how intelligent systems operate in real world.

One significant challenge CL faces is the *catastrophic forgetting*, *i.e.*, knowledge learned in the past fading away as the model continues to learn new tasks [15]. Among prior approaches to addressing this issue, episodic memory (EM) is one of the most effective approaches [3, 6, 7, 14, 16]. The basic idea of EM is to maintain a buffer that stores samples collected in the past and replays them periodically while training models with new samples. EM needs to have a sufficiently large capacity to achieve a desired accuracy, and such capacity in need may vary significantly since incoming data may contain a varying number of tasks and classes at different time slots and geo-locations [2]. Although not discussed in prior EM studies as they are mostly simulation-based, *memory* is obviously the first choice to implement EM due to fast access (50–150 ns) allowing timely model updates. However, in practice, the size of EM must be quite small, bounded by *limited* on-device memory capacity, especially on mobile and IoT devices. The limited EM size makes it difficult to scale up to a large number of tasks, preventing CL models from achieving high accuracy as training moves forward. Alternatively, EM can be located in large *storage* to store a significantly large number of old samples and use them for greatly improving model accuracy. However, the storage incurs slow access (25–250 μs), which is typically orders of magnitude larger than the memory. Hence, directly accessing the storage often incurs significant I/O overhead, which slows down training.

We introduce a hierarchical EM method, which enhances the effectiveness of *in-memory* EM design. Our method is motivated by the fact that modern computing devices are commonly equipped with a deep memory hierarchy including both fast memory and large storage. Provided by different hardware characteristics, our method 1) accesses samples *only* in memory during training, promising high-speed training, and 2) simultaneously leverages on-device storage to exercise a large number of old samples, addressing the forgetting problem. The goal of our work, **Carousel Memory** or **CarM**, is to design a CL framework enhanced with 1) and 2), making a step forward in bringing prior simulation-based studies to practical system setups.

CarM stores as many observed samples as possible so long as it does not exceed a given storage capacity and updates the in-memory EM in the background while the model is learning with samples already in EM. One key research question is how to manage samples across EM and storage for both system efficiency and model accuracy. Here we propose a hierarchical memory-aware *data swapping*, an *online* process that dynamically replaces a subset of in-memory samples used for model training with other samples stored in storage. The data swapping significantly improves the effective EM size, mitigating forgetting issues by avoiding discarding important samples overflowed from EM in limited size (as done in existing single-level non-hierarchical EM methods). To fulfill such effectiveness, we design CarM from two perspectives: **swapping mechanism** (Section 3.1) and **swapping policy** (Section 3.2). The swapping mechanism of CarM ensures that the slow speed

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530587

of accessing the storage does not become a bottleneck of continual model training by carefully hiding sample swapping latency through *asynchrony*. Moreover, we propose a swapping policy to select samples to swap and incorporate it into the *gate function*. The gate function selects a small set of important samples to swap, making CarM match with low I/O bandwidth storage.

One major benefit of CarM is that it is largely complementary to existing episodic memory-based CL methods. By exploiting the memory hierarchy, we show that CarM helps boost the accuracy of many existing methods and even allows them to retain their accuracy with much smaller EM sizes (Section 4.1).

Interestingly, we find that some algorithmic optimizations may need to be revisited to ensure that they are not actually at odds with data swapping. We observe that knowledge distillation with CarM might not be as effective as in prior work, because CarM opts for direct training with explicit data rather than probability to learn representation for old classes.

2 BACKGROUND AND RELATED WORK

Class incremental learning. The performance of CL algorithms heavily depends on scenarios and setups, as summarized by Van de Ven *et al.* [19]. Among them, we are particularly interested in class-incremental learning (CIL), where task IDs are not given during inference [10]. CIL targets more challenging yet realistic CL scenarios than the alternative approach called task-incremental learning (TIL), which assumes task IDs are given during inference. Many prior proposals in CIL are broadly divided into two categories, rehearsal-based and regularization-based. In rehearsal-based approaches, episodic memory stores a few samples of old tasks to replay in the future [2, 4, 5, 17]. On the contrary, regularization-based approaches exploit the information of old tasks implicitly retained in the model parameters, without storing samples representing old tasks [12, 13, 22]. As rehearsal-based approaches generally have shown the better performance in CIL [16], we aim to improve the approaches by incorporating data management across the memory-storage hierarchy.

The CIL setup usually assumes that the tasks contain disjoint set of classes [4, 10, 17]. More recent studies introduce methods to learn from the blurry stream of tasks, where some samples across the tasks overlap in terms of class ID [1, 16]. Moreover, prior works can be classified as either *offline* [4, 5, 17, 21], which allows a buffer to store incoming samples for the current task, or *online* [1, 9, 11], which has no such buffer. The data management scheme that we propose in CarM is applicable to all above setups and methods.

Rehearsal-based CIL problem. To illustrate the rehearsal-based CIL problem, assume we have seen i tasks $\{T_1, \ldots, T_i\}$ so far. A task T includes a subset of all classes C, denoted as $C(T) = \{c | \mho(c) = T\}$, where $\mho(c)$ performs a task assignment for a given class c. Further, the T includes a finite number of samples x and corresponding labels y. The EM, given a limited capacity, maintains a subset of old samples from previous tasks $\{T_1, \ldots, T_{i-1}\}$. CarM additionally maintains a larger pool of samples from the previous tasks in the storage ES, capturing past experiences more accurately—ES is inclusive of EM. However, since old samples for training are still drawn directly from in-memory EM, the training phase is forced to have a *boundary* for sample selection restricted by the size of EM. The

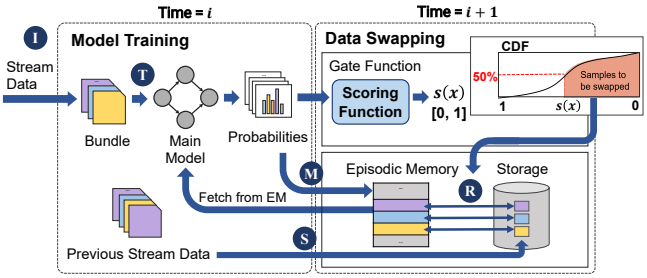


Figure 1: Architecture and execution stages of the CarM.

continual learning that optimizes model parameters θ for old and new samples can hence to formulated as follows:

$$\underset{\theta}{\operatorname{argmin}} \sum_{task\ id=1}^{i} E_{(x,y)\sim ES \cup T_i}[L(f(x,\theta),y)], \text{where } (x,y) \in EM.$$

3 CAROUSEL MEMORY

We describe how data swapping in CarM extends the current workflow of episodic memory (EM) in Figure 1. There are three common stages involved in traditional EM methods, which proceed in order: **data incoming**, **training**, and **EM updating**. This workflow corresponds to many popular non-hierarchical EM methods including TinyER [7], CBRS [8], iCaRL [17], BiC [21], and DER++ [3]. Then, we add two additional key stages for data swapping: **storage updating** and **storage sample retrieving**.

- **Data incoming** (I): When samples for a new task T_i arrive, they are first enqueued into a *stream buffer* and later exercised for training. Different CL algorithms require different amounts of samples to be enqueued for training. The *task-level* learning relies on task boundaries, waiting until all T_i's samples appear [17, 21]. On the contrary, the *batch-level* learning initiates the training stage once a batch of T_i's samples in a pre-defined size is available [7, 18].

- **Training** (T): The training combines old samples in EM with new samples in a stream buffer to compose a *bundle* of training data. Typically, in the task-level learning, a bundle is large and hence partitioned into multiple mini-batches, whereas in the batch-level learning, a bundle is small and hence constitutes itself a single mini-batch. The mini-batch size and the ratio between old and new samples within a mini-batch are configured by the learning algorithm. Learners may take multiple epochs of training for a given bundle, trading off computation cost for accuracy.

- **EM updating** (M): Once the training stage is completed, samples in the stream buffer are no longer new and represent a past experience, requiring EM to be updated with these samples. EM may have enough space to store all of them, but if it does not, the CL method applies a sampling strategy like *reservoir sampling* [20] and *greedy-balancing sampling* [16] to select a subset of samples from EM as well as from the stream buffer to keep in EM. Prior works "discard" samples that overflow from the EM capacity.

- **Storage updating** (S): CarM flushes the stream data onto the storage before cleaning up the stream buffer for the next data incoming step. Therefore, data samples are preserved in storage as long as the storage still has free space. However, if the storage capacity is exceeded, we randomly choose samples to evict for each class while keeping the in-storage data class-balanced.

• **Storage sample retrieving** (R): With a large number of samples maintained in the storage, data swapping replaces in-memory samples with in-storage ones during training to exercise abundant past knowledge preserved in the storage. CarM collects various useful signals for each in-memory sample used in the training stage and determines whether to replace that sample or not. This decision is made by our generic *gating function* that selects the samples for replacement with effectively little runtime cost.

3.1 Minimizing Delay to Continual Learning

Accessing the slow storage could incur high I/O latency and slow down the training process. The primary objective in our proposed system design is thus effectively hiding I/O latency caused by the data swapping so that CarM does not degrade the training speed. In this section, we introduce two system optimizations that encompass in-storage sample retrieval and EM updating stages.

Asynchronous sample retrieval. Similar to the conventional learning practice, CarM maintains fetch workers dedicated to pre-processing each training sample, *e.g.*, decoding and augmentation. CarM has a separate *swap worker* dedicated to deciding in-memory samples to evict and issuing I/O requests to bring new samples from storage into memory. In the CL workflow, the data retrieval stage R has dependency on the training stage T since training triggers the replacement of an in-memory sample when it is used as training input. To illustrate, let us assume that the system creates N mini-batches from a bundle. The swap worker identifies samples in EM to be replaced from mini-batch i training (T_i^b) and then issues I/O requests to retrieve other samples from storage (R_i^b). If we want to allow the next mini-batch training to exercise EM completely refreshed with the replaced samples, executions of T^b and R^b by definition must be serialized such that we have a sequence of $T_1^b \rightarrow R_1^b \rightarrow T_2^b \rightarrow R_2^b \rightarrow T_3^b \rightarrow R_3^b$, as shown in Figure 2 (Sync). However, committing to such strict serialized executions slows down training speed significantly, *e.g.*, the second mini-batch training T_2^b can start only after finishing up $T_1^b \rightarrow R_1^b$, which takes much longer time than T_1^b with no retrieval of storage data as done in the traditional EM design. To prevent this performance degradation, CarM adopts *asynchronous sample retrieval* that runs the retrieval step *in parallel* with the subsequent training steps. By the asynchronous method, we keep the minimum possible dependency as shown in Figure 2 (Async), with an arbitrary R_i^b not necessarily happened before T_{i+1}^b. Apparently, this design choice poses a delay on applying in-storage samples to EM, making it possible for the next training steps to access some samples undergoing replacement. However, we found such accesses do not frequently occur, and the delay does not nullify the benefit that comes from data swapping.

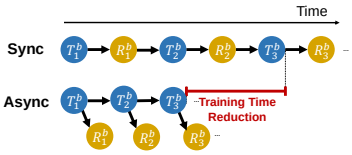


Figure 2: Training time reduction with async sample retrieval.

Concurrency on EM. In addition, when the swap worker retrieves in-storage samples and writes on memory, it may interfere with fetch workers that attempt to read samples from it for pre-processing. To mitigate such interference, CarM could opt for EM partitioning to parallelize read/write operations over independent partitions. With EM partitioning, only those operations that access the same partition need coordination, achieving concurrency against operations that access other partitions.

3.2 Data Swapping Policy by a Gate Function

The gate function in Figure 1 allows CarM to select a certain portion of samples to swap out from those EM samples drawn in the training stage, providing a control knob for I/O traffic. The ability of adjusting I/O traffic is crucial as on-device storage mediums have different characteristics (*e.g.*, high-bandwidth flash drive *vs* low-bandwidth magnetic drive). At the same time, the gate is required to be effective with such partial data swapping in terms of achieving high accuracy in the subsequent training steps. To facilitate this, we propose a *sample scoring* method that ranks the samples in the same mini-batch and helps decide samples to swap against other samples to keep further in memory.

Score-based replacement. The score quantifies the relative importance of a trained sample to keep in EM with respect to other samples in the same mini-batch. Intuitively, a higher score means that the sample is better "not" to be replaced if we need to reduce I/O traffic and vice versa. To this end, we define the gate σ_i as $\sigma_i = \mathbb{1}(s(x_i) > \tau)$ for sample x_i, where $s(\cdot)$ is a scoring function and τ is a scoring threshold, with both $s(\cdot)$ and τ between 0 and 1. The threshold is determined by the proportion of the samples in a mini-batch to be replaced from the EM. It allows data swapping to match with I/O bandwidth available on the storage medium, and prevents the system from over-subscribing the bandwidth leading to I/O back-pressure or under-subscribing the bandwidth leaving storage data exploited less opportunistically.

Policy. From several swapping policies we have explored, here we introduce an entropy-based policy. This policy collects two useful signals for each sample produced during training: prediction correctness and the associated entropy. It prefers to replace the samples that are correctly predicted because these samples may not be much beneficial to improve the model in the near future. Furthermore, in this group of samples, if any specific sample has a lower entropy value than the other samples, the prediction confidence is relatively stronger, making it a better candidate for replacement. By contrast, for the samples that are incorrectly predicted, this policy prefers to "not" replace the samples that exhibit lower entropy, *i.e.*, incorrect prediction with stronger confidence, since they may take longer to be predicted correctly. Thus, the scoring function $s(x_i)$ with a model $f(\cdot)$ is defined as:

$$s(x_i) = \frac{1}{2u}\left[g(x_i)H(f(x_i)) + (1 - g(x_i))\,(2u - H(f(x_i)))\right],$$

where $g(x_i) = \mathbb{1}(f(x_i) = y_i)$, $H(\cdot)$ is an entropy function, and u is the maximum entropy value. Note that we replace an EM sample with an in-storage sample with the same class label to keep the population in EM consistent across classes.

We highlight that although our major contribution for gating mechanism is computational efficiency to match with available on-device I/O bandwidth, designing more effective replacement policies is a promising research direction that we are currently exploring.

3.3 Impact of CarM on CL Algorithms Using Knowledge Distillation

CarM is largely complementary to the existing methods and can help improve their model accuracy by exploiting abundant in-storage samples in the background. However, some algorithmic features may not be compatible with our data-driven approach, requiring their effectiveness under CarM to be revisited. We discuss this aspect in the context of knowledge distillation (KD), which has been adopted in several popular CL algorithms [3, 17, 21].

The key assumption of KD is that once a model is trained with a task, the knowledge newly learned is supposed to generalize the task well and can be effectively transferable to subsequent task training. Specifically, during a model update with a new task, KD methods exploit soft labels (*i.e.*, classification probabilities over possible classes) obtained from the previous model while replaying old data. As recent KD methods (*e.g.*, BiC [21] and DER++ [3]) further combine soft labels and hard labels (*i.e.*, ground truth) to update the model for old data, the loss function of distillation-based methods can be generally written as "$\alpha \times soft\ label\ loss + \beta \times hard\ label\ loss$", where α and β are the weight coefficient on the corresponding loss term. Note that the soft label loss is also called the *distillation loss*.

The potential drawback of using KD is that old models might not be sufficiently generalized for old tasks. In this case, data swapping in CarM can be used to further generalize old tasks by continuously exercising abundant in-storage samples. However, data swapping can be interfered by the knowledge distilled by the old models if distillation losses are extensively used, *i.e.*, training the current model still heavily relies on the performance of the previous model. In prior studies [3, 17, 21], the weight coefficient α on the distillation loss is either high or managed complicatedly. In Section 4.1, we show that limiting α to a small value delivers higher accuracy with CarM for those methods.

3.4 Implementation Details

CarM is implemented on PyTorch for ease of use.

Swap worker. CarM implements the swap worker through multi-processing in popular Python standard library. The swap worker uses *asyncio* to asynchronously load samples from storage to memory, effectively overlapping high-latency I/O operations with other CarM-related operations, such as image decoding, sample replacement on EM, and entropy calculation. The swap worker issues multiple data swapping requests without spinning on or being blocked by I/O. As a result, it is sufficient to have only one swap worker for CarM in the system.

Episodic memory. There are various ways to implement EM to be shared between fetch workers and the swap worker. Currently, we opt for implementing EM as a shared object provided by *Manager* in the Python standard library (*multiprocessing.managers*), which is based on message passing in the server-client semantics. In terms of flexibility, the Manager does not require the clients (*i.e.*, fetch workers and the swap worker) to define the exact data layout in the EM address space or coordinate for potential memory resizing to accommodate raw samples of different sizes. Hence, it is sufficient for the client workers to perform reads and writes on the EM using indexes on the EM samples.

4 EXPERIMENTS

Experimental setup. We evaluate CarM on two computing devices. `Device 1` is a server equipped with a NVIDIA 2080 Ti GPU, Intel Xeon Gold 6226 12 cores, and a 480 GB Intel SSD D3 drive. `Device 2` is a NVIDIA Jetson TX2 equipped with 256-core NVIDIA Pascal GPU, Denver 64-bit 2 cores, ARM Cortex-A57 4 cores, and a 32 GB eMMC 5.1 drive. We use Device 1 for most of experiments to evaluate CarM while varying memory sizes in use for episodic memory. We use Device 2 to show CarM's efficacy when running on a real embedded AI computing device.

We compare CarM with three non-hierarchical EM setups: 1) **RAM-S** manages old samples in RAM-based EM with limited size; 2) **ST-S** manages old samples directly in storage, using the same capacity for EM used by RAM-S; and 3) **ST-L** manages old samples in storage without the capacity constraint. **CarM** implements data swapping on top of RAM-S. For CarM, we vary the amount of data swapping to study the effectiveness of CarM in detail. Unless otherwise stated, **CarM-N** means that our swap worker is configured to replace N% of EM samples drawn by the training stage.

Workloads. CarM is applicable to a variety of EM-based CL methods. We select three methods that achieved the state-of-the-art accuracy: **BiC** [21], **DER++** [3], and **RM** [2]. BiC and DER++ are knowledge distillation-based methods. BiC and RM correspond to the task-level learning (that initiates training at a task boundary), whereas DER++ corresponds to the batch-level learning (that initiates training upon receiving a batch of new samples). These methods are based on either ResNet or DenseNet neural networks, with all using the SGD optimizer.

We cover a wide range of datasets including **CIFAR subset**—CIFAR10 (**C10**/196.6MB) and CIFAR100 (**C100**/196.5MB)—, **ImageNet subset**—ImageNet-100 (**I100**/46.8GB) and Tiny-ImageNet (**TI200**/1.1GB)—, and **ImageNet-1000** (**I1000**/139.3GB). EM sizes tested in our experiments range from 3 MB to 13GB to represent diverse memory specifications. Note that to keep the original effectiveness of the selected methods, we measure the performance with and without CarM in the methods of their own setups as used in the original works, *e.g.*, dataset, EM size, hyper-parameters, etc.

Metrics. We use the **final accuracy** averaged over classes to reflect the performance of continual learning. Except for ImageNet-1000 that represents a significantly large-scale training, the final accuracy is averaged over five runs. We also measure **training time** by referring to the wall-clock time (i.e., actual time taken) of the end-to-end continual learning.

4.1 Results

Comparison to non-hierarchical EM designs. We compare RAM-S, ST-S, and ST-L with CarM that performs full swapping (*i.e.*, CarM-100). Figure 3 presents the final accuracy using the top-1 accuracy, except for BiC that measures the top-5 accuracy for the ImageNet subset as done in [21]. Figure 4 reports the training time normalized by RAM-S. We push data into the stream buffer at a rate enough to keep training always busy with new mini-batches. Note that RAM-S, ST-S, and CarM use the EM with limited size, as shown in the second item in the parenthesis of the figures for each CL method.

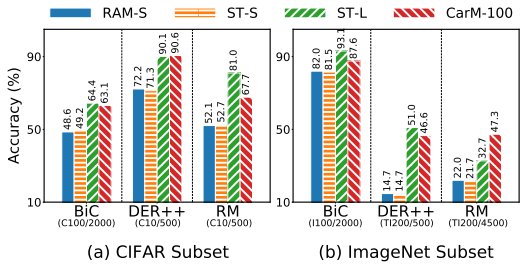


Figure 3: The final accuracy of competing EM setups. For each CL method, we show the dataset and EM size in bytes in the parenthesis.

Method	BiC	DER++	RM
Async	-0.3%/+1.7%	+0.3%/+2.4%	+2.3%/-0.9%
Sync	+20.0%/+33.8%	+71.6%/+38.8%	+25.9%/+2.6%

Table 1: Training time for CarM-50 w.r.t. RAM-S. (+) training time increases, (-) training time decreases. (/) results for the CIFAR subset and the ImageNet subset.

First, as Figure 3 shows, CarM improves the accuracy remarkably over RAM-S that uses only in-memory EM, advancing the state-of-the-art performances for CIFAR and ImageNet datasets through data swapping. The results clearly show the effectiveness of using the storage device in large capacity along with memory to allow CL to exploit abundant information of the previous tasks. The accuracy in ST-S is comparable to RAM-S as it uses in-storage EM of the same size. Obviously, ST-L delivers relatively higher accuracy than CarM as it does not have any capacity limit in EM. However, ST-L has to entail excessive direct accesses to the in-storage samples to deliver greater accuracy gains, thereby having to incur significantly large operational costs during training, as shown in Figure 4.

Figure 4 shows the training time normalized by RAM-S. We observe negligible slowdowns with CarM for all CL methods due to delay optimization techniques such as asynchronous sample retrieval. Despite having comparable accuracy to RAM-S, ST-S takes 52% more training time than RAM-S, confirming that memory is more desirable than storage for the same EM size. ST-L has a similar level of slowdown to ST-S in DER++, since DER++ is the batch-level learning where both ST-S and ST-L retrieve the same predefined number of old samples from storage upon receiving a batch of new samples. However, for BiC and RM, which are the task-level learning, ST-L has to combine a new task along with all samples in EM to compose a bundle of training data. In this case, ST-L shows the slowdown up to 15.21×, making it unfavorable to pursue if fast and efficient on-device training is the key design objective. Figure 4 shows the results when using 5–10 tasks as done in the original works. We observe the slowdown is much worse when we increase the number of tasks into 100 for the datasets.

Async *vs* Sync. The asynchronous sample retrieval (**Async**) presented in Section 3.1 incurs insignificant delay on training as shown in Figure 4. Here we examine how training speed in CarM changes over RAM-S as applied with synchronous sample retrieval (**Sync**). As shown in Table 1, the synchronous version slows down training time up to 71.6% for CIFAR subset and 38.8% for ImageNet subset.

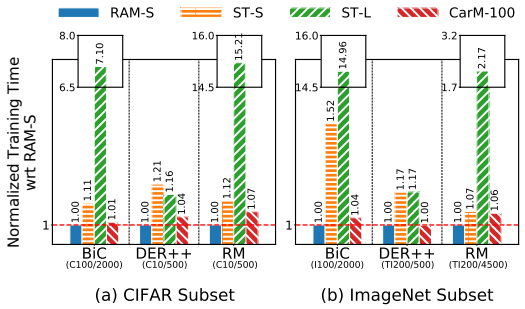


Figure 4: Normalized training time with respect to RAM-S.

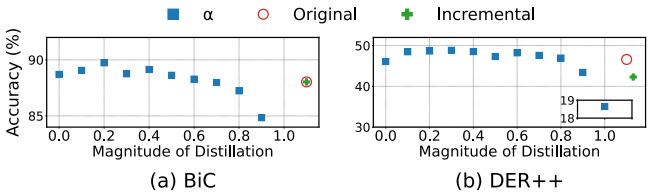


Figure 5: Accuracy of BiC and DER++ with CarM-50 on ImageNet subset while varying α values on the distillation loss.

CarM on knowledge distillation. As discussed in Section 3.3, we can further increase accuracy of knowledge distillation-based CL methods (*i.e.*, BiC and DER++) under CarM by adjusting how aggressively the distillation loss is used. To this end, using the ImageNet subset, we show accuracy over varying the weight coefficient α on the distillation loss (or soft label loss) in Figure 5. The weight coefficient β on the hard label loss is set to be $1 - \alpha$ to reflect its relative level of consideration while varying α values. For each method, we also include accuracy when α increases incrementally as training proceeds with more tasks, as done in BiC [21].

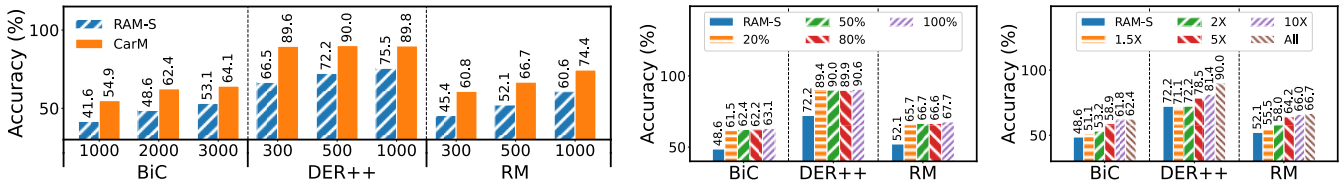
The results show that the distillation-based methods with CarM significantly improve accuracy when the α is very small. In particular, for BiC, compared to $\alpha = 0.9$, we obtain 3.9% higher accuracy when $\alpha = 0.0$ (*i.e.*, no distillation) and 4.9% higher accuracy when $\alpha = 0.2$, which is the best result. Moreover, with CarM, the coefficient does not necessarily be managed complicatedly to achieve higher accuracy, as done in the original BiC (*i.e.*, `Incremental`).

4.2 Additional Study

We present an ablation study (Figure 6) using the CIFAR subset while comparing CarM to RAM-S. We also show the effectiveness of CarM under a large-scale training using ImageNet-1000 dataset (Figure 7) or on NVIDIA Jetson TX2 (Figure 8).

Size of EM. To confirm performance benefits over using different memory sizes, we empirically evaluate CarM-50 over varying EM sizes and show the average accuracy in Figure 6(a). In all cases, CarM-50 considerably outperforms RAM-S for all the three CL methods. Moreover, we observe that data swapping delivers better accuracy over the memory-only approaches using much smaller memory. For example, CarM-50 with DER++ on EM size 300 shows higher accuracy than pure DER++ on EM size 1000.

Data swapping ratio. We present results with different swapping ratios (Figure 6(b)) to show that our gate model indeed brings



(a) Episodic Memory Sizes (b) Data Swapping Ratios (c) Storage Capacities

Figure 6: Accuracy over varying (a) EM sizes, (b) data swapping ratios, and (c) storage capacities.

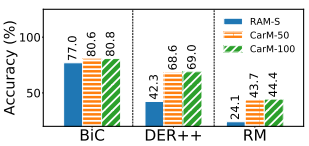


Figure 7: CarM accuracy with ImageNet-1000 dataset.

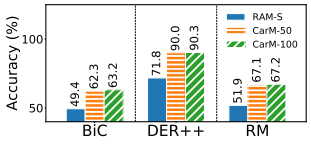


Figure 8: CarM accuracy on NVIDIA Jetson TX2 (CIFAR subset).

out meaningful benefits over using different I/O bandwidths. A surprising result is that even at CarM-20 in 20% data swapping, the accuracy is very comparable to when we allow higher data swapping ratios. The result indicates that our system would be effective even when applied to the storage with low bandwidth.

Size of storage. As local storage cannot store all the past data, the system must discard some old samples once the storage is fully occupied. Figure 6(c) shows accuracy degradation in CarM-50 when storage capacity is limited to 1.5–10× of the EM size. The results show that data swapping improves performance over traditional approaches even with using 50% larger capacity for the storage.

Large number of tasks. One pressing issue on CL is to effectively enable a large-scale continual learning as it is required to keep the knowledge learned in the remote past. To evaluate this aspect, we split ImageNet-1000 (1000 classes) into 10 tasks, each with 100 classes, and run with the three CL methods. As Figure 7 shows, CarM still outperforms RAM-S at a large gap, showing the potential for long-term continual model training.

On an embedded device. Figure 8 shows the efficacy of CarM when running on an embedded device. We see similar trends to Figure 3 for CarM's consistently higher accuracy than RAM-S and to Figure 6(b) for CarM's effectiveness with using a smaller data swapping ratio. Moreover, we observe that the absolute accuracy of RAM-S, CarM-50, and CarM-100 does not degrade due to using a device with lower computing capacities (as compared to Figure 3).

5 CONCLUSION

We alleviate catastrophic forgetting by integrating episodic memory-based continual learning methods with device-internal data storage, named CarM. We design data swapping strategies to improve model accuracy by dynamically utilizing a large amount of the past data available in the storage. Our swapping mechanism addresses the cumbersome performance hurdle incurred by slow storage access, and hence continual model training is not dramatically affected by data transfers between memory and storage. We show the efficacy of CarM using three well-known methods on standard datasets.

ACKNOWLEDGMENTS

This work was partly supported by Electronics and Telecommunications Research Institute(ETRI) grant (22ZS1300), the National Research Foundation of Korea(NRF) grant (NRF-2021R1F1A1063262), IITP grant No.2020-0-01361-003 (AI Graduate School-Yonsei Univ.) and 2021-0-02068 (AI Innovation Hub), and Rebellions Inc.

REFERENCES

[1] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. 2019. Gradient Based Sample Selection for Online Continual Learning. In *NeurIPS*.
[2] Jihwan Bang, Heesu Kim, YoungJoon Yoo, Jung-Woo Ha, and Jonghyun Choi. 2021. Rainbow Memory: Continual Learning with a Memory of Diverse Samples. In *CVPR*.
[3] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and SIMONE CALDERARA. 2020. Dark Experience for General Continual Learning: a Strong, Simple Baseline. In *NeurIPS*.
[4] Francisco M. Castro, Manuel J. Marin-Jimenez, Nicolas Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-End Incremental Learning. In *ECCV*.
[5] Arslan Chaudhry, Puneet K. Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. 2018. Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence. In *ECCV*.
[6] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2019. Efficient Lifelong Learning with A-GEM. In *ICLR*.
[7] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. 2019. On Tiny Episodic Memories in Continual Learning. *arXiv:1902.10486* (2019).
[8] Aristotelis Chrysakis and Marie-Francine Moens. 2020. Online Continual Learning from Imbalanced Data. In *ICML*.
[9] Enrico Fini, Stéphane Lathuilière, Enver Sangineto, Moin Nabi, and Elisa Ricci. 2020. Online Continual Learning under Extreme Memory Constraints. In *ECCV*.
[10] Alexander Gepperth and Barbara Hammer. 2016. Incremental Learning Algorithms and Applications. In *ESANN*.
[11] Xisen Jin, Junyi Du, and Xiang Ren. 2020. Gradient Based Memory Editing for Task-Free Continual Learning. *arXiv:2006.15294* (2020).
[12] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming Catastrophic Forgetting in Neural Networks. In *PNAS*.
[13] Zhizhong Li and Derek Hoiem. 2017. Learning without Forgetting. In *IEEE Trans. PAMI*.
[14] David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient Episodic Memory for Continual Learning. In *NeurIPS*.
[15] M. McCloskey and Neal. 1989. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In *Psychology of Learning and Motivation*.
[16] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. 2020. GDumb: A Simple Approach that Questions Our Progress in Continual Learning. In *ECCV*.
[17] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. 2017. iCaRL: Incremental Classifier and Representation Learning. In *CVPR*.
[18] Dongsub Shim, Zheda Mai, Jihwan Jeong, Scott Sanner, Hyunwoo Kim, and Jongseong Jang. 2021. Online Class-Incremental Continual Learning with Adversarial Shapley Value. In *AAAI*.
[19] Gido M van de Ven and Andreas S Tolias. 2018. Three Continual Learning Scenarios and a Case for Generative Replay. In *NeurIPS Workshop on Continual Learning*.
[20] Jeffrey S Vitter. 1985. Random Sampling with a Reservoir. In *ACM TOMS*.
[21] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. 2019. Large Scale Incremental Learning. In *CVPR*.
[22] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual Learning Through Synaptic Intelligence. In *ICML*.