Testing the limits of data efficiency in experience replay

Damiano Meier[1], **Giulia Lanzillotta**[1,2], and **Thomas Hofmann**[1]

[1] Department of Computer Science, ETH Zurich
[2]ETH AI Center

Abstract

In Continual Learning rehearsal-based methods store a subset of observed data in a buffer for replay during training. The computational efficiency of these methods is tied to their data efficiency, i.e., the size of their buffer. In this work we expose a nuanced picture of rehearsal, underscoring the role of implicit biases on the road towards scalable CL.

1 Introduction

Deep learning has proved successful in solving several hard benchmarks, often leaving researchers puzzled on its surprising efficacy. One of the deficits of the learning paradigm, which is currently limiting further progress, is the structural inability to learn continually. It is widely documented that updating the model with new, potentially different information may completely erase the knowledge acquired through training, an effect termed *catastrophic forgetting* (CF) [6]. A simple solution, which has been widely adopted in practice is to store all the new samples and regularly retrain the model from scratch. However, with the current trend towards bigger models and higher training computational costs, this solutions becomes impracticable. The goal of *continual learning* (CL) is to modify the learning algorithm for neural networks in order to incorporate data sequentially, without interference.

Rehearsal based methods are among the first procedures developed to reduce CF [9]. These methods operate by storing a small subset of the observed data in a *buffer*, which is then *replayed* when training on new data. These methods have remained extremely popular to this day due to their simplicity and consistently strong performance. However, their effectiveness depends on making efficient use of the buffer, as its size directly impacts computational overhead and storage requirements. Several rehearsal based algorithms have been developed over the years, including a broad variety of techniques to enhance the buffer's efficiency. Among the computationally cheapest ones is using *knowledge distillation* (KD) [2].

In this paper, we investigate the flaws of existing rehearsal methods using KD, and exploit this knowledge to maximise their efficiency. Focusing on data efficiency is key to reduce the need for large replay buffers, enabling more scalable continual learning solutions. We discover that the quality of logits plays a critical role in the effectiveness of distillation based rehearsal methods. We identify both temporal and structural biases in the teacher model's logits, which can undermine the performance of KD-based rehearsal. Finally, we expose that the scale of the model has a crucial impact on buffer storage efficiency.

2 Background & Notation

We look at supervised continual learning, where each task consists in predicting classification labels $y \in \mathcal{Y} = [K]$ from an input $x \in \mathcal{X} \subseteq \mathbb{R}^d$. We denote by $z = h_\theta(x)$ the log-probabilities or *logits* for a model h parameterized by θ and an input x, and by $f(x) = \sigma(z)$ the associated predictor -

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

with optional normalization σ. The algorithm is presented with a sequence of M tasks, with datasets $D_1, \ldots, D_M$. We focus on *task-incremental* (TIL) and *class-incremental* (CIL) learning, where each task adds a new set of classes. The -crucial- difference between these two settings is the softmax normalization, which can include only the current task's classes (TIL) or all the classes (CIL) [10].

The simplest form of rehearsal is **Experience Replay** (ER) [9], which consists in storing a subset $S_\tau = \{(x_i^\tau, y_i^\tau)\}_{i=1}^{N_\tau} \subset D_\tau$ of N_τ samples from every task τ, and training the model on $S_1 \cup \cdots \cup S_{t-1} \cup D_t$. Despite its simplicity, ER works very well on most benchmarks (even beyond supervised CL). However its efficacy depends on the buffer size N_τ and therefore most of the research on rehearsal has focused on reducing N_τ without sacrificing performance. Knowledge distillation has achieved some success in this regard. Among the most renowned methods using KD-based rehearsal there is **Learning without Forgetting** (LwF) [5], which stores a smoothed version of the current network's outputs for new samples before training on them, using KD to minimize their drift during the training process. **iCaRL** [8] uses a buffer to approximate the features class means for each observed class, and also uses the network before training as a teacher on the buffer examples. **Dark experience Replay** (DER) applies KD using the model outputs *while training* on a task as soft targets during replay [2]. Surprisingly, DER achieves striking performance improvements on all baselines, effectively accomplishing SoTA on KD-based rehearsal. In the remainder of this paper we choose ER and DER as baselines for their simplicity, which makes them more suitable for study, and their effectiveness (see Appendix B for additional background on ER and DER).

3 Logits

Undoubtedly, the central element in KD is the logits provided by the teacher. We thus take this as a starting point to examine the efficiency of distillation-based rehearsal, compared to naive rehearsal.

High quality logits improve efficiency

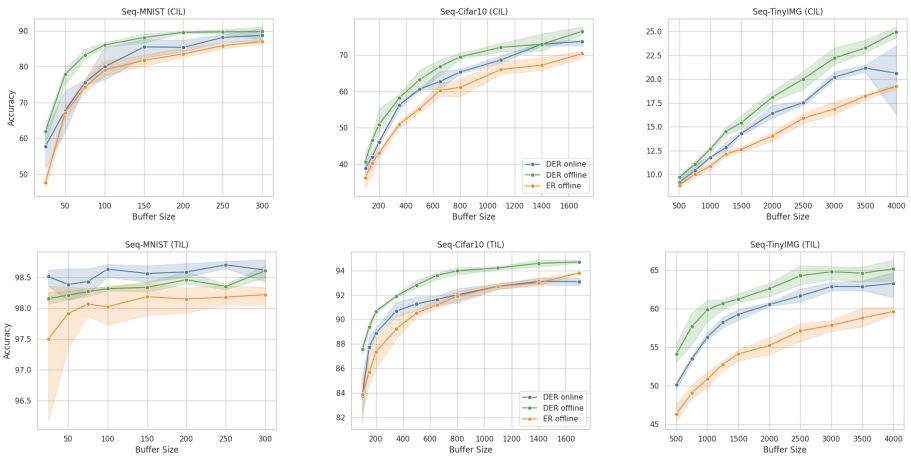

Figure 1: **Sampling strategy matters.** Comparing the classification results of online and offline sampling as a function of the buffer size, averaged over 3 seeds. The top row represents results using the CIL training paradigm, while the bottom row corresponds to the TIL paradigm.

Intuitively, a model with better performance should yield better logits. Therefore logits collected later in training should be of higher "quality". But what is the impact of the "quality" of the logits on the replay data efficiency? We answer this question by modifying the buffer sampling frequency of DER. Instead of sampling regularly throughout training (online sampling) we sample only at the end of each task (offline sampling). Notice that at the end of each task both strategies contain an equal fraction of samples per task in expectation. However, during the task training, the online buffer contains strictly less samples from the previous tasks as demonstrated in Figure 9. Importantly, the models accuracy consistently improves throughout training, and therefore the accuracy of the logits in the offline buffer is always strictly higher than those in the online buffer. In Figure 11 we show that the average accuracy of the buffer logits can be up to 10% lower with the online sampling strategy.

In Figure 1 we compare online and offline sampling as a function of the buffer size. As initially suspected, offline sampling generally results in a better performance, and higher data efficiency. This evidence seems to confirm the importance of high-quality logits in the buffer.
Additionally, following previous work which has highlighted the role of temperature in KD, we evaluate offline sampling with a fine-tuned temperature on the dataset.[1] In Figure 10 we show the effect of temperature on the efficiency.

Overall, with optimally-tuned temperature and high quality logits, KD-rehearsal shows markedly higher efficiency in the task-incremental setting, reaching the same performance of ER with an even 4 times smaller buffer. However, the benefits of distillation-based replay are not as pronounced for the class-incremental setting. This raises the question of whether there is something wrong with self-distillation in a CIL setting.

Temporal and structural bias in class-incremental learning As briefly mentioned above, the crucial difference between the CIL and TIL settings lies in the normalization. Let C be the number of new classes introduced by each task, then the TIL and CIL predictors lie respectively in the C-dimensional and $C \times M$-dimensional simplex.

Distillation based rehearsal methods typically appoint a previous version of the model as a teacher. Assuming all the classes are equally present in the dataset and in the buffer, the mean estimator $\mathbb{E}_{y \in Y_{\text{train}}}[y]$ in the TIL setting is the uniform distribution $\rho(c) = \frac{1}{C} \, \forall \, c \in [C]$ for all the observed tasks, whereas the mean estimator in the CIL setting evolves with the number of tasks as $\rho_\tau(c) = \frac{1}{\tau \cdot C}$, effectively downplaying the frequency of the future tasks' classes. Moreover, if the number of samples from the current task is over-represented during training, the mean estimator will have a bias to the current task, assigning a higher probability to the classes in the current task. We call this bias a *temporal bias* in CIL.

Additionally, we identify a *structural bias* in the evolving covariance of the teacher's predictions. According to one of the most prevalent theories of distillation -the *dark knowledge hypothesis*- the benefit of KD over hard label targets must reside in the class relationship structure implicit in a teacher network's output. The sequentially trained teacher used in distillation-based rehearsal may develop a different covariance than a model trained jointly on all the tasks.

With the following experiments we evaluate the impact of temporal and structural bias on the performance of distillation based rehearsal in the CIL setting. First, we employ a teacher jointly trained on all tasks, which we call *JTTeacher*.[2] Clearly, the JTTeacher model has no bias. In order to evaluate the impact of the structural and temporal biases we compare it to the offline DER baseline and two additional predictors, namely *'JTTeacher + permuting'* and *'Label Smoothing'*. The former consists in applying a random permutation to the teacher output, which keeps only the true class fixed and destroys the class relationship structure encoded in the second order moments of the original distribution. Label smoothing instead consists in increasing the entropy of the label targets distribution by adding uniform noise to all the classes. Both predictors have the wrong structural bias, since their covariance has a flat structure. However while the entropy of label smoothing is constant over the data points, the permuted predictor entropy is the same as the original teacher's.

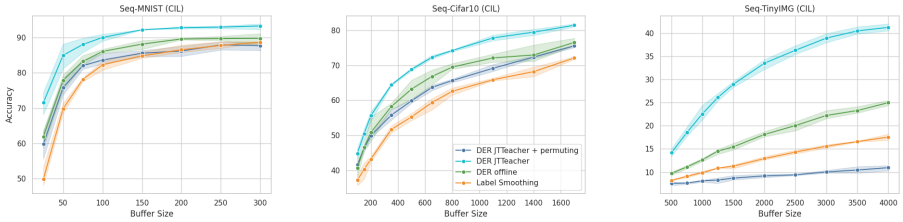

Figure 2: **Using a jointly trained teacher leads to substantial performance gains.** The plot shows classification results as a function of buffer size when employing different predictors for buffer sampling.

[1] It can be shown [4] that the MSE gradient is equivalent to the KL gradient in the infinite temperature limit.

[2] Note that this model would not be admissible in a CL benchmark where data is not available to the algorithm before it is observed. However, we use it here as an experimental tool which allows us to test the hypotheses.

In Figure 2 we see that indeed KD rehearsal with the JTTeacher displays the best performance, with especially remarkable gains in Tiny Imagenet. This results shows that logits bias has a key role in the low performance of KD in CIL. Label Smoothing and JTTeacher + permuting achieve even worse results than the baseline, suggesting that a non-informative structure in the outputs is detrimental to data efficiency in KD. Surprisingly, we find that the temporal bias does not play a crucial role in the performance of the predictors. In Figure 12 we observe that the DER predictor has a strong bias to the current task, and that removing this bias (as in the case of JTTeacher + permuting or Label Smoothing) is not enough to achieve good performance in CIL. Additionally, we observe that the temporal bias is stronger for lower buffers.

Taken together, these results suggest that achieving better efficiency in distillation-based rehearsal depends fundamentally on having a good teacher, with the right structural and temporal bias. In Appendix D we investigate whether this bias propagates back to the feature layer or if it is fundamentally a prediction head problem. We find that a similar temporal bias can be observed at the feature level and yet that the feature classification accuracy is relatively stable when decreasing buffer sizes, which suggests that the bottleneck in data efficiency is the linear layer.

Bias in batch norm statistics. The over-representation of the current task has a direct effect on the batch norm statistics, which are updated through a running mean while training. In particular, during evaluation, the moments μ and σ^2 are set to a past-discounting running average of the mean and variance from recent mini-batches. The standard implementation of rehearsal involves two forward and backward passes in each training step: one using a mini batch from the current task and another using samples from the buffer. If the mini batches are of the same size, this inevitably biases the estimation of μ and σ^2 towards the current task. In Appendix C we perform a simple experiment to ascertain the presence of bias in batch norm.

4 Scaling

Finally, we explore the effects of scaling the backbone network. In deep learning, larger models are often able to achieve better performance. However, even small networks tend to overfit the replay buffer, which leads to catastrophic forgetting. It is thus unclear whether the same improvements can be observed in the buffer efficiency curves as we scale the networks up.

We first evaluate ResNet18 (11M), ResNet34 (21M), and ResNet50 (24M) [3] on two challenges—Seq-CIFAR and TinyIMG—under both CIL and TIL settings. We find (Table 6) that increasing the model size results in a decline in performance, with the smallest model, ResNet18, outperforming the larger counterparts. The fact that all networks always fit the training dataset nearly perfectly suggests that, as expected, the larger models might be more prone to overfitting. Additionally, in terms of data efficiency, we see that increasing the model size does not lead to consistent improvements in the buffer utilization: the performance increment with x3 buffer size is approximately constant across model sizes.

Next, we decide to simplify the network architecture and scale vanilla CNN models (details in Appendix A.1) with layer norm by increasing the number of channels. Surprisingly we observe a different picture when scaling the CNN. As illustrated in Figure 13, increasing the model size consistently improves performance. The largest model generally achieves the best classification accuracy and displays the lowest forgetting values. Importantly, we observe better buffer utilization in larger networks, particularly on Tiny ImageNet, where the benefits of a larger buffer were more pronounced for larger networks. Even when scaling the CNN up to 50M and 100M parameters, the performance showed slight improvements or at least remained constant.

5 Conclusion

In summary, we have looked at the data efficiency of KD-based rehearsal, showing that, implicit temporal and structural biases in the teaching signal limit data efficiency beyond logits quality. Moreover, we study the effect of implicit bias in architectural components such as batch normalization and overfitting and model scale. Overall, our findings suggest that progress towards scalable solutions in CL depends on better inductive biases rather than model scaling on its own.

References

[1] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

[2] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. CALDERARA. Dark experience for general continual learning: a strong, simple baseline. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15920–15930. Curran Associates, Inc., 2020.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[4] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015.

[5] Z. Li and D. Hoiem. Learning without forgetting, 2017.

[6] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[7] Q. Pham, C. Liu, and S. Hoi. Continual normalization: Rethinking batch normalization for online continual learning, 2022.

[8] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning, 2017.

[9] A. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[10] G. M. van de Ven and A. S. Tolias. Three scenarios for continual learning, 2019.

Appendices

A Experimental setup

A.1 Training Framework

We employed the benchmark codebase developed by [2][3]. In Table 1 we summarize the configurations used in the main experiments. The seeds were fixed to [1000, 2000, 3000] and all displayed results are aggregated acoress those 3 seeds. To obtain the jointly trained teacher (JTTeacher) the seed was fixed to 1000. We performed to notable modifications to the existing codebase.

Dataset	Challenge type	Tasks	Network
Seq-MNIST	CIL, TIL	5	MLP (89K)
Seq-Cifar10	CIL, TIL	5	ResNet18 (11M)
Seq-TinyIMG	CIL, TIL	10	ResNet18 (11M)

Table 1: **Experiment configurations.**

First, due to the issues explained in Appendix C, we have avoided using BN layers to remove any confounding factors associated with them. Therefore, unless noted otherwise, all BN layers were replaced by LN [1] layers. Table 3 provides a comparison of ER and DER trained on ResNet18 with either BN or LN layers in the CIL paradigm. As other works [7] have pointed out, we have observed that BN has positive effect on forgetting and thus achieves a better overall performance.

Second, the existing codebase doesn't distinguish between CIL and TIL training. In both cases, the softmax normalization includes all classes seen so far. It only differentiates between them when evaluating the model, as in TIL only the classes of the current task are considered. We followed what is described in Section 2 and have adapted the codebase accordingly.

For Section 4, we used a CNN with four convolutional layers. Each layer applies a 3x3 convolution filter, followed by LN and a ReLU activation function. At each convolutional layer, we double the number of channels. The second and third layer also include a 2x2 max-pooling layer at the end, and an adaptive max-pooling layer follows the fourth convolutional layer to reduce the height and width to one. Finally, the network concludes with a fully connected classification layer. We evaluated the following six size configurations where we scale the model by increasing the number of channels in the first convolutional layer:

Network	Parameters	Channels in first layer
CNN1	100K	16
CNN2	400K	32
CNN3	1.6M	64
CNN4	4.8M	112
CNN5	11.7M	176
CNN6	21.8M	240

Table 2: **CNN sizes.**

Regarding the training we followed [2] by training all networks with Stochastic Gradient Descent (SGD) and a constant learning rate. We have not used weight decay or momentum. Seq-MNIST was trained for 1 epoch, Seq-Cifar10 for 50 epochs, and Seq-TinyIMG for 100 epochs. In contrast, the pretrained model was trained for 20 epochs on MNIST, 100 epochs on Cifar10, and 150 epochs on Tiny-ImageNet to account for the slower convergence when training on the joint distribution.

A.2 Hyperparameters

Since we modified the networks, we had to retune the hyper-parameters. To that end, we initially conducted a preliminary search to identify the most promising hyper-parameter combinations for

[3]Their codebase is publicaly availible at: https://github.com/aimagelab/mammoth

Buffer	Network	Seq-Cifar10 (CIL), ACC(↑)		Seq-Cifar10 (CIL), FGT(↓)	
		ResNet18 + BN	ResNet18 + LN	ResNet18 + BN	ResNet18 + LN
200	ER online	47.24 ± 2.25	40.45 ± 1.92	60.65 ± 4.50	68.13 ± 2.64
	DER online	59.06 ± 2.68	46.06 ± 0.63	43.70 ± 4.27	61.90 ± 0.81
500	ER online	53.94 ± 2.19	53.16 ± 0.34	53.19 ± 2.64	51.88 ± 0.39
	DER online	70.24 ± 1.04	60.62 ± 0.25	28.77 ± 2.04	43.26 ± 0.74
1400	ER online	62.70 ± 2.26	66.46 ± 0.52	42.73 ± 2.66	35.75 ± 1.02
	DER online	74.44 ± 3.66	72.95 ± 0.48	23.44 ± 5.38	26.89 ± 1.71

Buffer	Network	Seq-TinyIMG (CIL), ACC(↑)		Seq-TinyIMG (CIL), FGT(↓)	
		ResNet18 + BN	ResNet18 + LN	ResNet18 + BN	ResNet18 + LN
500	ER online	10.15 ± 0.28	8.78 ± 0.10	74.85 ± 0.64	64.33 ± 1.14
	DER online	18.72 ± 0.27	9.16 ± 0.38	63.23 ± 1.00	64.415 ± 0.23
1500	ER online	16.13 ± 0.11	12.28 ± 0.42	68.39 ± 0.39	58.77 ± 2.25
	DER online	26.39 ± 1.08	14.32 ± 0.23	50.19 ± 1.64	57.76 ± 0.54
4000	ER online	24.71 ± 0.64	18.32 ± 0.73	57.60 ± 0.70	52.54 ± 0.39
	DER online	33.67 ± 1.02	20.64 ± 3.64	40.20 ± 1.67	48.10 ± 3.36

Table 3: **BN reduces forgetting.** Classification results on standard CL benchmarks using a ResNet18 backbone with either BN orLN layers. Results are averaged across 3 different seeds.

each dataset. The results were obtained by considering the hyper-parameters listed in Table 4, from which we selected and displayed the best outcomes. We used the same set of hyper-parameters for all DER based algorithms and likewise for all ER based algorithms.

Regarding the temperature T used for softmax normalization, we introduced it as a tunable hyper-parameter for DER offline. To approximate the infinite temperature case of the KL loss, we utilized the MSE loss to avoid numerical instabilities. However, for DER online we adhered to the original approach proposed by [2], consistently using the MSE loss.

Dataset	Network	ER + variations	DER + variations
Seq-MNIST (CIL)	MLP	$lr : [0.1, 0.01]$	$lr : [0.1, 0.03]$, $\alpha : [0.5, 1.0, 2.0]$, $T : [mse]$
Seq-MNIST (TIL)	MLP	$lr : [0.1, 0.01]$	$lr : [0.1, 0.03]$, $\alpha : [0.5, 1.0, 2.0]$, $T : [20]$
Seq-Cifar10 (CIL)	ResNet18 + LN	$lr : [0.03]$	$lr : [0.01]$, $\alpha : [0.03, 0.1]$, $T : [20]$
Seq-Cifar10 (TIL)	ResNet18 + LN	$lr : [0.03]$	$lr : [0.01]$, $\alpha : [0.03, 0.1]$, $T : [20]$
Seq-Cifar10 (CIL)	ResNet34 + LN	-	$lr : [0.01]$, $\alpha : [0.03, 0.1]$, $T : [20]$
Seq-Cifar10 (TIL)	ResNet34 + LN	-	$lr : [0.01]$, $\alpha : [0.03, 0.1]$, $Te : [20]$
Seq-Cifar10 (CIL)	ResNet50 + LN	-	$lr : [0.01, 0.005]$, $\alpha : [0.03, 0.1]$, $T : [20]$
Seq-Cifar10 (TIL)	ResNet50 + LN	-	$lr : [0.01, 0.005]$, $\alpha : [0.03, 0.1]$, $T : [20]$
Seq-Cifar10 (CIL)	ResNet18 + BN	$lr : [0.1]$	$lr : [0.03]$, $\alpha : [0.3]$
Seq-Cifar10 (CIL)	CNN (1 - 6)	-	$lr : [0.01]$, $\alpha : [0.03, 0.1]$, $T : [20]$
Seq-Cifar10 (TIL)	CNN (1 - 6)	-	$lr : [0.01]$, $\alpha : [0.03, 0.1]$, $T : [20]$
Seq-TinyIMG (CIL)	ResNet18 + LN	$lr : [0.01]$	$lr : [0.01]$, $\alpha : [0.5]$, $T : [mse]$
Seq-TinyIMG (TIL)	ResNet18 + LN	$lr : [0.01]$	$lr : [0.01]$, $\alpha : [0.5]$, $T : [20, mse]$
Seq-TinyIMG (CIL)	ResNet34 + LN	-	$lr : [0.005]$, $\alpha : [0.1, 0.5]$, $T : [mse]$
Seq-TinyIMG (TIL)	ResNet34 + LN	-	$lr : [0.005]$, $\alpha : [0.1, 0.5]$, $T : [20, mse]$
Seq-TinyIMG (CIL)	ResNet50 + LN	-	$lr : [0.01]$, $\alpha : [0.1, 0.5]$, $T : [mse]$
Seq-TinyIMG (TIL)	ResNet50 + LN	-	$lr : [0.01]$, $\alpha : [0.1, 0.5]$, $T : [20, mse]$
Seq-TinyIMG (CIL)	ResNet18 + BN	$lr : [0.1]$	$lr : [0.03]$, $\alpha : [0.1]$
Seq-TinyIMG (CIL), buffer size: [500, 1500]	CNN (1 - 4)	-	$lr : [0.01]$, $\alpha : [0.1]$, $T : [mse]$
Seq-TinyIMG (TIL), buffer size: [500, 1500]	CNN (1 - 4)	-	$lr : [0.01]$, $\alpha : [0.1]$, $T : [20]$
Seq-TinyIMG (CIL), buffer size: [4000]	CNN (1 - 4)	-	$lr : [0.03]$, $\alpha : [0.2]$, $T : [mse]$
Seq-TinyIMG (TIL), buffer size: [4000]	CNN (1 - 4)	-	$lr : [0.03]$, $\alpha : [0.2]$, $T : [20]$
Seq-TinyIMG (CIL), buffer size: [500, 1500]	CNN (5, 6)	-	$lr : [0.01, 0.003]$, $\alpha : [0.1]$, $T : [mse]$
Seq-TinyIMG (TIL), buffer size: [500, 1500]	CNN (5, 6)	-	$lr : [0.01, 0.003]$, $\alpha : [0.1]$, $T : [20]$
Seq-TinyIMG (CIL), buffer size: [4000]	CNN (5, 6)	-	$lr : [0.03, 0.01]$, $\alpha : [0.2]$, $T : [mse]$
Seq-TinyIMG (TIL), buffer size: [4000]	CNN (5, 6)	-	$lr : [0.03, 0.01]$, $\alpha : [0.2]$, $T : [20]$

Table 4: **Hyperparameters.**

A.3 Details on online and offline sampling algorithms

For the online sampling strategy, we used reservoir sampling as described in [2]. This approach ensures that every sample from the input stream has an equal probability of being stored, making it compatible with a more general understanding of CL that does not depend on tasks boundaries during

training. However, it only ensures an equal number of samples for each task in expectation by the end of each task.

On the other hand, our offline sampling strategy guarantees that the replay buffer contains an equal number of samples (up to remainders) for each task. After completing training on a task, the strategy removes samples from previous tasks proportionally and randomly selects samples from the current task to fill the buffer. It is important to note that this sampling method requires task boundaries during training.

A.4 Additional details on the experiments

Table 5 shows the buffer sizes which were used to obtain Figure 12 and Figure 7. The large buffer size was used to obtain Figure 11. For Figure 6 the small buffer size was used. The same behavior can be observed with other buffer sizes, where larger buffer sizes generally display a slightly higher classification accuracy for the previous tasks.

Dataset	small buffer	large buffer
Seq-MNIST	50	250
Seq-Cifar10	200	1100
Seq-TinyIMG	1000	30000

Table 5: **Buffer sizes.**

Figure 3 was created with Seq-Cifar10. However, we have observed the same behavior in Seq-Cifar100 and Seq-TinyIMG.

Label Smoothing in Figure 2 is a modified version of ER offline where the label smoothing parameter of the cross entropy loss was set to 0.1.

B Background ER and DER

B.1 Experience Replay

On of the earliest algorithms developed to address CF is ER [9]. The core idea behind ER is simple: a small memory buffer is maintained, containing examples from past tasks. When the model trains on a new task, mini-batches of data are drawn not only from the current task but also from the stored buffer. This simultaneous exposure to both old and new data forces the optimization process to balance the acquisition of new information with the preservation of old knowledge.

The goal is to learn how to correctly classify, at any point in training, examples from any of the observed tasks $t \in \{1, ..., M\}$:

$$\underset{\theta}{\operatorname{argmin}} \sum_{t=1}^{M} \mathcal{L}_t, \quad where \quad \mathcal{L}_t \coloneqq \mathbb{E}_{(y,x)\sim D_t}[l(y, f_\theta(x))]. \tag{1}$$

Directly solving this is challenging, as the previous tasks are assumed unavailable. Therefore the best θ for $\mathcal{L}_{1:M}$ must be found without having access to D_t for $t \in \{1, ..., M-1\}$. To overcomes this limitation, ER introduces a replay buffer $\mathcal{R}_t$, storing data samples of past tasks t. The replay buffer can be used to approximate the optimization problem of Equation (1):

$$\underset{\theta}{\operatorname{argmin}} \ \mathcal{L}_M + \sum_{t=1}^{M-1} \mathbb{E}_{(y,x)\sim \mathcal{R}_t}[l(y, f_\theta(x))]. \tag{2}$$

The quality of this approximation crucially depends on the size of the replay buffer. In case of an infinite buffer size, both optimization objectives are identical.

In practice, ER is typically implemented such that for every training mini-batch from the current task, a mini-batch of the same size is drawn uniformly at random from the replay buffer. These two mini-batches are then concatenated for the forward and backward passes. As the mini-batches from the replay buffer are randomly sampled, the optimization objective can be rewritten to:

$$\underset{\theta}{\operatorname{argmin}} \ \mathcal{L}_{t_c} + \mathbb{E}_{(y,x)\sim \mathcal{R}}[l(y, f_\theta(x))]. \tag{3}$$

B.2 Dark Experience Replay

DER was originally introduced by Buzzega et al. in [2] as a simple yet highly effective enhancement to the ER baseline. While ER stores the input x and corresponding label y in the replay buffer, DER makes a key modification: it stores the input x along with the logits z. Leveraging these stored logits during replay helps to preserve more information about past tasks, which aids in mitigating CF.

The main objective remains the same as established in Equation (1). Now the idea of DER is to look for parameters that fit the current task well while approximating the original responses for past tasks. It therefore seeks to minimize the following objective:

$$\underset{\theta}{\operatorname{argmin}} \ \mathcal{L}_M + \alpha \sum_{t=1}^{M-1} \mathbb{E}_{(x)\sim D_t}[D_{KL}(f_{\theta_t^*}(x) \| f_\theta(x))] \tag{4}$$

where $f_{\theta_t^*}(x)$ is the optimal set of parameters at the end of task t and α is a hyper-parameter balancing the trade-off between the terms. This objective, which resembles the teacher-student approach, requires access to datasets D_t of previous tasks. In the same manner as ER, DER introduces a replay buffer $\mathcal{R}$ to overcome this limitation. But instead of storing the ground truth labels y, DER retains the network logits z along with the input x in $\mathcal{R}$:

$$\underset{\theta}{\operatorname{argmin}} \ \mathcal{L}_{t_c} + \alpha \sum_{t=1}^{M-1} \mathbb{E}_{(z,x)\sim \mathcal{R}_t}[D_{KL}(\sigma(z) \| f_\theta(x))]. \tag{5}$$

The authors have decided to opt for matching logits, as under mild assumptions [4], the optimization of the KL divergence is equivalent to minimizing the Euclidean distance between the corresponding pre-softmax responses (i.e. logits). With these considerations in hands, DER optimizes the following objective:

$$\underset{\theta}{\operatorname{argmin}} \ \mathcal{L}_{t_c} + \alpha \sum_{t=1}^{M-1} \mathbb{E}_{(z,x)\sim \mathcal{R}_t}[\|z - f_\theta(x)\|_2^2]. \tag{6}$$

In practice, DER is typically implemented such that for every training mini-batch from the current task, a mini-batch of the same size is drawn uniformly at random from the replay buffer. Then the algorithm does two separate foreward and backward passes and the gradients from both backward passes are summed for the optimization step. Because of randomly sampling from the replay buffer, the optimization formula can be rewritten to:

$$\underset{\theta}{\operatorname{argmin}} \ \mathcal{L}_{t_c} + \alpha \mathbb{E}_{(z,x)\sim \mathcal{R}}[\|z - f_\theta(x)\|_2^2] \tag{7}$$

C Batch Normalization biased statistics

A Batch Normalization (BN) layer operates on a mini-batch of feature maps $\mathcal{B} = a_1, ..., a_n$. Each feature map $a_k \in \mathbb{R}^{D_k}$ is centered and normalized: $a'_k = \gamma(\frac{a_k - \mu}{\sqrt{\sigma^2 + \epsilon}}) + \beta$. γ and β are learnable parameters to keep the layers representation capacity and ϵ is a small constant to avoid division by zero. The moments μ and σ^2 are computed differently in *training* or *evaluation* mode. In training mode, μ and σ^2 are the mean and variance of the mini batch. In evaluation mode μ and σ^2 are a past-discounting running average of the mean and variance from recent mini-batches.

The standard implementation of rehearsal involves two forward and backward passes in each training step: one using a mini-batch from the current task and another using samples from the buffer. When using offline sampling the buffer does not include samples from the current task τ. As a result, the BN statistics and parameters, used in the forward passes with samples from the current task, do not match the aggregated statistics across all observed tasks $[1, \tau]$ which leads to poor performance on the current task during evaluation, as shown in Figure 3. Simply using moments calculated across the mini-batch dimension for evaluation resolves the poor performance of the current task.

This discovery suggests that the effectiveness of online sampling observed in [2] is partially due to the implicit bias of BN. In fact, when comparing offline and online sampling in BN-free architectures (Figure 1) we fail to make the same observation.

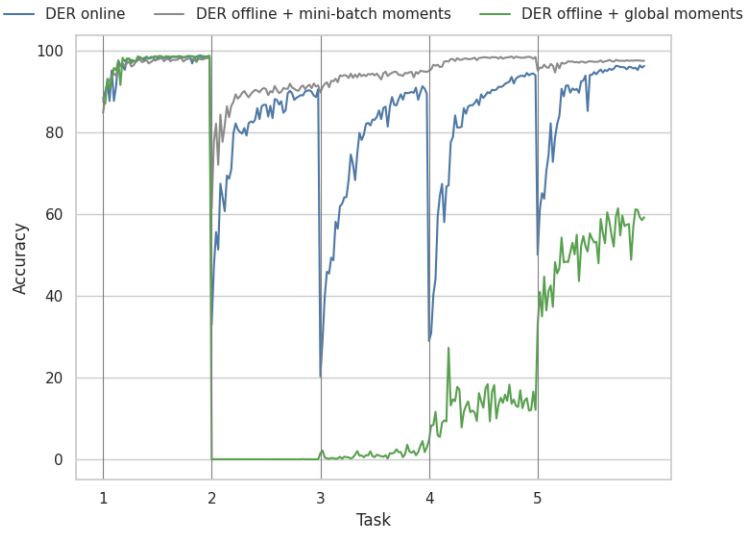

Figure 3: **Impact of BN global estimates on classification performance.** The figure shows the evaluation accuracy of DER offline after every epoch (each task is trained for 50 epochs) in ResNet18 with BN layers. The evaluation accuracy is compared using either mini-batch moments or their corresponding global estimates for normalization.

D Features

In this chapter, we analyze the network's feature layer to extend the results from the previous chapter. To assess the quality of the learned features, we replace the original output layer with one that has been trained on the complete dataset, composed of the datasets of all previously seen tasks. This approach provides an optimal decision boundary based on the features, at least with respect to the training dataset, enabling us to distinguish between forgetting at the feature level and forgetting due to a sub optimal output layer. Figure 4 presents a comparison between the classification results discussed in Section 3 and those obtained using the fitted output layer. As shown, using the fitted head consistently improves classification accuracy over the original network head.

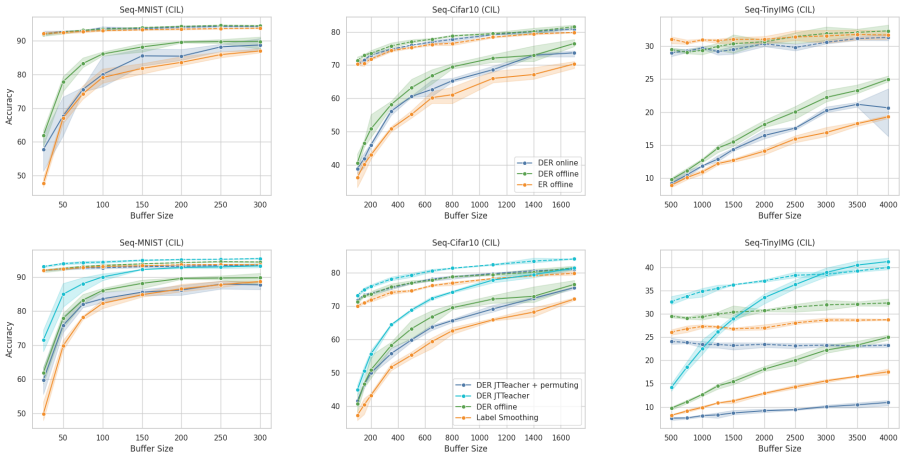

Figure 4: **A substantial portion of forgetting is attributed to the output layer.** The plot presents classification results when evaluating the model using either the fitted output layer (dashed line) or the original network head (solid line) as a function of the buffer size, using different predictors for buffer sampling. Mean and average are aggregated over 3 seeds.

Although the absolute accuracy values are higher when using the fitted head, the relative ranking of the algorithms remains somewhat stable. In the top row, we observe that at the feature layer, the differences between ER and DER, with an online or offline sampling strategy, are less significant. All three algorithms enable the network to learn comparable qualitative features. However, in the bottom row, the differences at feature-level become more pronounced. As shown, using JTTeacher for KD allows the network to learn superior features compared to the self-distillation methods. Furthermore, we see that predictors with an incorrect structural bias, such as Label Smoothing, negatively impact feature learning and result in worse features.

Finally, we observe that refitting the output layer results in significant performance improvements, particularly for smaller buffer sizes. In fact, it allows smaller buffers to outperform even the largest buffer with the original network head. Additionally, contrary to what is generally observed when evaluating the model with the network head, the classification performance remains nearly constant when increasing the buffer size.

To determine whether this improvement stems from reduced forgetting, we further analyze the forgetting metrics in the next section.

D.1 Feature forgetting

In Figure 5 we compare the forgetting values when evaluating the model using either the original network head or the fitted head. We can observed that the fitted head displays extremely low forgetting values which remain nearly constant when increasing the buffer size. While the overall classification accuracy is indeed higher with the fitted head, as was shown in the previous section, this alone does not account for the significant reduction in forgetting.

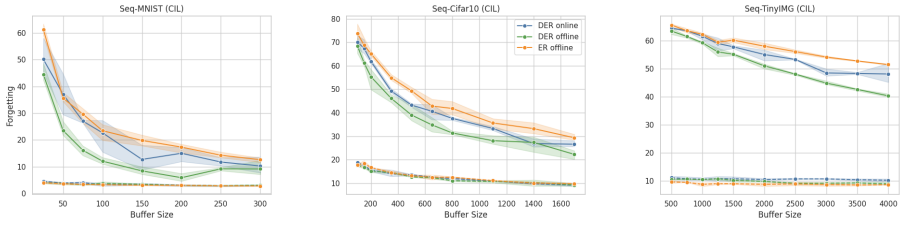

Figure 5: **A substantial portion of forgetting is attributed to the output layer.** Comparison of forgetting values when evaluating the model using either the network head (solid line) or the fitted head (dashed line) for online and offline sampling, shown as a function of the buffer size. The results are averaged over 3 seeds.

Figure 6 compares the mean classification accuracy of the previous tasks to the classification accuracy of the current task. As discussed in Section 3, the model predictions tend to be biased towards the current task, resulting in noticeably higher accuracy values for that task. However, with the fitted head, the accuracy on the current task decreases, resulting in a more balanced performance across all tasks. This balance implies the existence of a more optimal decision boundary that mitigates excessive bias towards the current task. This observation leads to a further question: does a similar temporal bias exist within the feature or deeper network layers, or is it unique to the output layer?

D.2 Temporal bias at the feature layer

To assess the informativeness of the features, we again utilize an output layer that has been trained using the complete dataset from all seen tasks. However, unlike in the previous section, here we fit a separate output layer for each task and evaluate each task separately. In essence, we train the model using the CIL paradigm but evaluate it under the TIL paradigm. The idea is to determine whether the model performs better on the current task compared to previous ones. If the classification performance is superior for the current task, we can infer that the features encode more information about the current task's samples, indicating a bias toward the current task.

In Figure 7, we compare the classification accuracy for the current task against the mean accuracy across previous tasks. Regardless of the buffer size, the features support better classification for the current task's classes, suggesting they carry more information about these classes and are thus biased

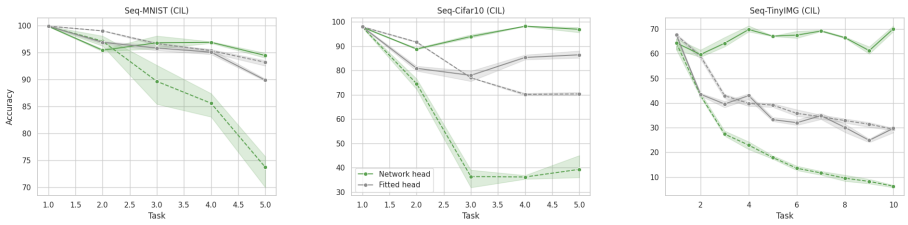

Figure 6: **Fitting the output layer reduces the bias towards the current task.** The plot shows the classification accuracy for the current task (solid line) compared to the mean classification accuracy of previous tasks (dashed line), for both the network head and the fitted head using DER offline. Results are averaged over three seeds.

towards the current task. Note how this temporal bias persists even when an infinite buffer is used. Only in MNIST we can observe inconclusive results. However, as the classification accuracy is nearly perfect in MNIST, those differences are probably attributed to irreducible noise.

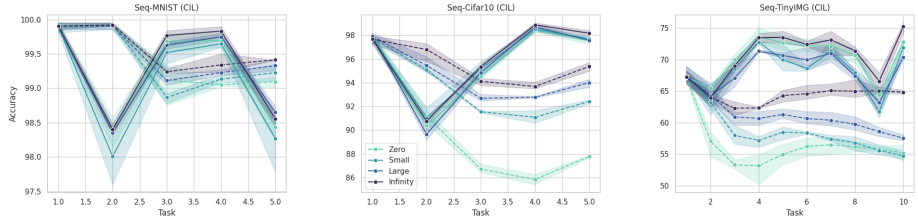

Figure 7: **Temporal bias persists at the feature layer.** The plot shows the classification accuracy for the current task (solid line) compared to the mean classification accuracy of previous tasks (dashed line) when separately fitting an output layer for every task. The results are shown for different buffer sizes using DER offline and averaged over three seeds.

Furthermore, when evaluating the feature quality for earlier tasks, we observe an improvement when transitioning from no buffer to an infinite buffer and the gap between previous task and current task is reduced. It is important to note that even without a buffer, the features retain some capacity to classify the old tasks' classes since the tasks come from the same dataset, making them somewhat related. However, when comparing performance between the small and large buffer size, the improvement for previous tasks is minimal, much less significant than the performance gain seen at the output layer. This is also reflected in Figure 4, where performance with the fitted head is nearly independent of the buffer size. It suggests that the relatively small buffers sizes (in comparison to the dataset size) typically used in rehersal-based methods primarily influence the output layer, with little effect on the feature layer.

These findings indicate that temporal bias is present not only in the output layer but also extends to deeper layers. Notably, the size of the replay buffer significantly impacts the output layer, with a lesser effect on the feature layer. Having access to a well-optimized output layer in CIL enables accurate and unbiased predictions even with minimal buffer sizes. Although this approach shifts the problem to the output layer rather than resolving it entirely, it does offer the advantage that the optimization problem at the output layer is convex.

E Buffer overfitting when scaling the network

In section Section 4, we observed that scaling up a vanilla CNN enhances overall performance and reduces catastrophic forgetting. Achieving this improvement requires not only enhancing performance on the current task but also maintaining or improving accuracy on previously learned tasks. This leads to a key question: how can larger networks, with over ten million parameters, avoid overfitting the limited buffer and outperform smaller networks when the buffer contains only a few hundred samples?

To investigate performance on prior tasks and determine if larger models are more prone to overfitting the buffer, we conduct the following experiment: before training on the final task we create a copy of the network. One version continues typical training on both the current task and the buffer, while the copy is trained exclusively on buffer samples. By comparing performance on prior tasks (excluding predictions for classes in the current task), we assessed whether larger models maintain or improve accuracy on previously learned tasks.

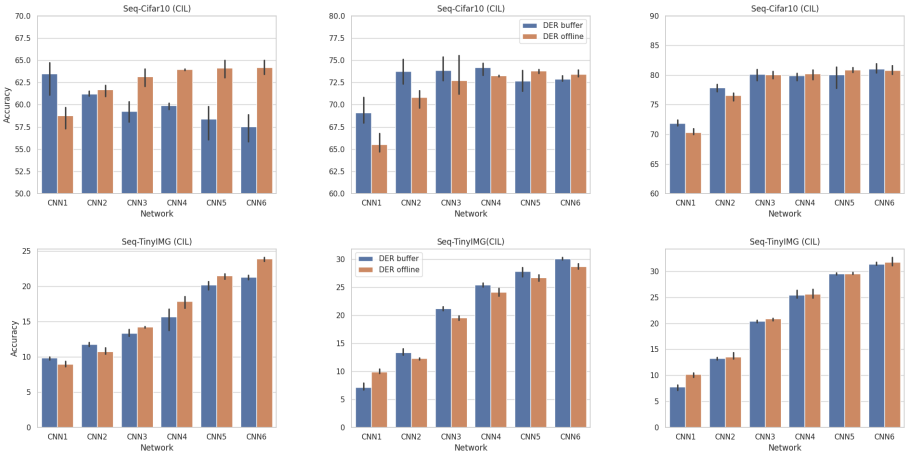

Figure 8: **Scaling networks does not increase buffer overfitting** The plot shows classification performance on previous tasks for networks trained in two conditions: DER offline and DER buffer, where for the last task the model trains exclusively on the buffer. Predictions for classes of the current task are ignored when evaluating the model. The buffer sizes used for the top row, from left to right, are 200, 500, and 1400 samples, while the bottom row displays results for buffer sizes of 500, 1500, and 4000 samples. Results are averaged over 3 seeds.

For CIFAR-10, we observe (Figure 8) that with buffer sizes of 200 and 500, the largest model did not achieve the best performance when trained solely on buffer samples. However, when trained on both the current task and the buffer, the largest model achieves the best performance. This suggests that for Cifar10 training on the current task helps to counteract overfitting. However, for TinyImage, we found that performance on prior tasks consistently improved, with the largest network achieving the best results in both cases.

F Additional Plots

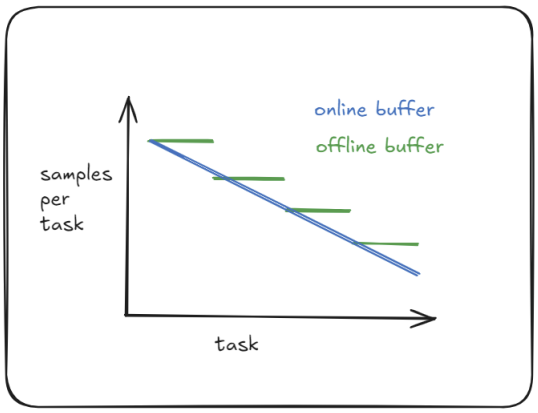

Figure 9: **The online buffer contains strictly less samples from previous tasks during training.** Sketch comparing the amount of samples from previous tasks present in the replay buffers when employing either an offline or online sampling strategy.

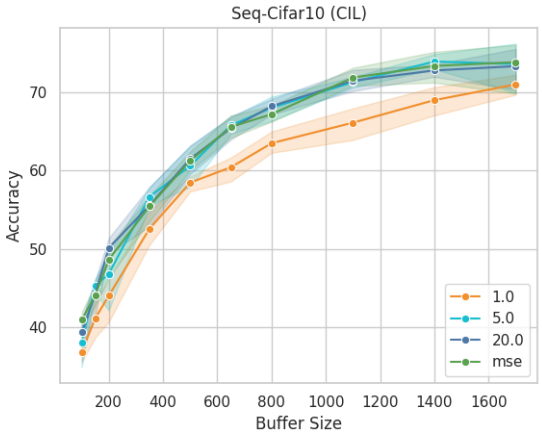

Figure 10: **Classification performance is not overly sensitive to the temperature.** The plot illustrates classification results as a function of the buffer size for DER offline when using different temperatures for normalization in the distillation process. Results are averaged over 3 seeds.

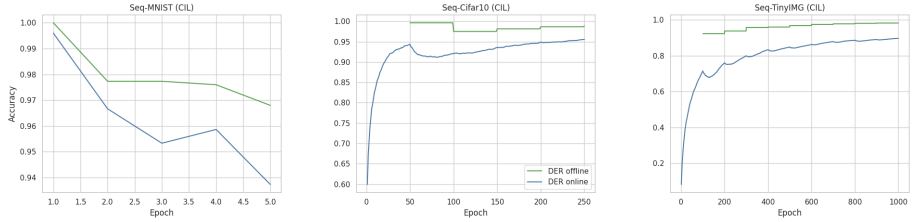

Figure 11: **Top-1 accuracy of buffer logits is consistently higher with offline sampling**. The plot displays the average top-1 accuracy of the buffer logits for both offline and online sampling strategies. Each task consists in respectively, from left to right, 1,50 and 100 epochs.

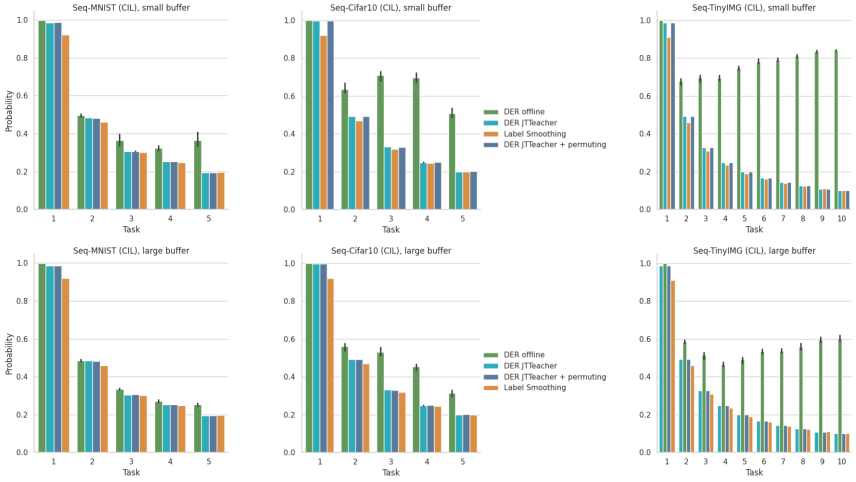

Figure 12: **DER predictor exhibits a temporal bias**. The figure illustrates the probability mass assigned to the current task by different predictors, assessed across samples of the observed classes for two different buffer sizes. Observe that with a larger buffer, the DER predictor demonstrates reduced temporal bias.

Buffer	Network	Seq-Cifar10, ACC(↑)		Seq-Cifar10, FGT(↓)	
		Class-IL	Task-IL	Class-IL	Task-IL
200	ResNet18	**50.86** $\pm$ 3.44	**90.66** $\pm$ 0.13	**55.34** $\pm$ 4.33	**6.35** $\pm$ 0.29
	ResNet34	45.67 $\pm$ 0.30	85.95 $\pm$ 1.30	62.30 $\pm$ 0.55	11.45 $\pm$ 0.86
	ResNet50	38.05 $\pm$ 4.36	82.50 $\pm$ 3.65	65.48 $\pm$ 3.26	11.43 $\pm$ 1.95
500	ResNet18	**63.23** $\pm$ 2.81	**92.84** $\pm$ 0.33	**39.02** $\pm$ 3.44	**3.65** $\pm$ 0.49
	ResNet34	59.46 $\pm$ 2.94	92.30 $\pm$ 0.55	43.39 $\pm$ 2.71	4.05 $\pm$ 0.85
	ResNet50	50.92 $\pm$ 4.60	87.16 $\pm$ 3.24	47.95 $\pm$ 3.44	5.21 $\pm$ 0.92
1400	ResNet18	**72.94** $\pm$ 2.34	**94.60** $\pm$ 0.29	**27.38** $\pm$ 3.19	**1.69** $\pm$ 0.29
	ResNet34	69.95 $\pm$ 4.37	94.42 $\pm$ 0.21	30.16 $\pm$ 5.57	1.71 $\pm$ 0.37
	ResNet50	61.97 $\pm$ 5.46	89.12 $\pm$ 2.12	33.02 $\pm$ 3.84	2.90 $\pm$ 1.24

Buffer	Network	Seq-TinyIMG, ACC(↑)		Seq-TinyIMG, FGT(↓)	
		Class-IL	Task-IL	Class-IL	Task-IL
500	ResNet18	9.73 $\pm$ 0.34	**54.10** $\pm$ 1.22	63.38 $\pm$ 1.10	14.52 $\pm$ 0.51
	ResNet34	**9.98** $\pm$ 0.63	51.26 $\pm$ 1.12	61.61 $\pm$ 0.21	14.12 $\pm$ 1.12
	ResNet50	9.30 $\pm$ 0.15	51.26 $\pm$ 2.14	**59.50** $\pm$ 0.07	**12.18** $\pm$ 1.67
1500	ResNet18	**15.45** $\pm$ 0.71	**61.23** $\pm$ 0.57	55.12 $\pm$ 0.43	6.34 $\pm$ 0.11
	ResNet34	15.26 $\pm$ 0.28	57.80 $\pm$ 1.15	53.84 $\pm$ 0.12	7.65 $\pm$ 0.69
	ResNet50	12.26 $\pm$ 0.19	57.82 $\pm$ 1.15	**51.11** $\pm$ 1.21	**5.22** $\pm$ 0.35
4000	ResNet18	**25.00** $\pm$ 0.56	**65.16** $\pm$ 1.36	40.33 $\pm$ 0.56	2.74 $\pm$ 0.21
	ResNet34	22.37 $\pm$ 0.60	61.35 $\pm$ 0.35	41.00 $\pm$ 0.89	**1.50** $\pm$ 0.14
	ResNet50	19.35 $\pm$ 0.48	59.60 $\pm$ 2.28	**40.02** $\pm$ 1.43	2.36 $\pm$ 0.37

Table 6: **Scaling ResNet doesn't improve the performance.** Classification results for DER offline on standard CL benchmarks when scaling the backbone network. Note that for ResNet50 the input was resized to (224,224). Results are averaged across 3 different seeds.

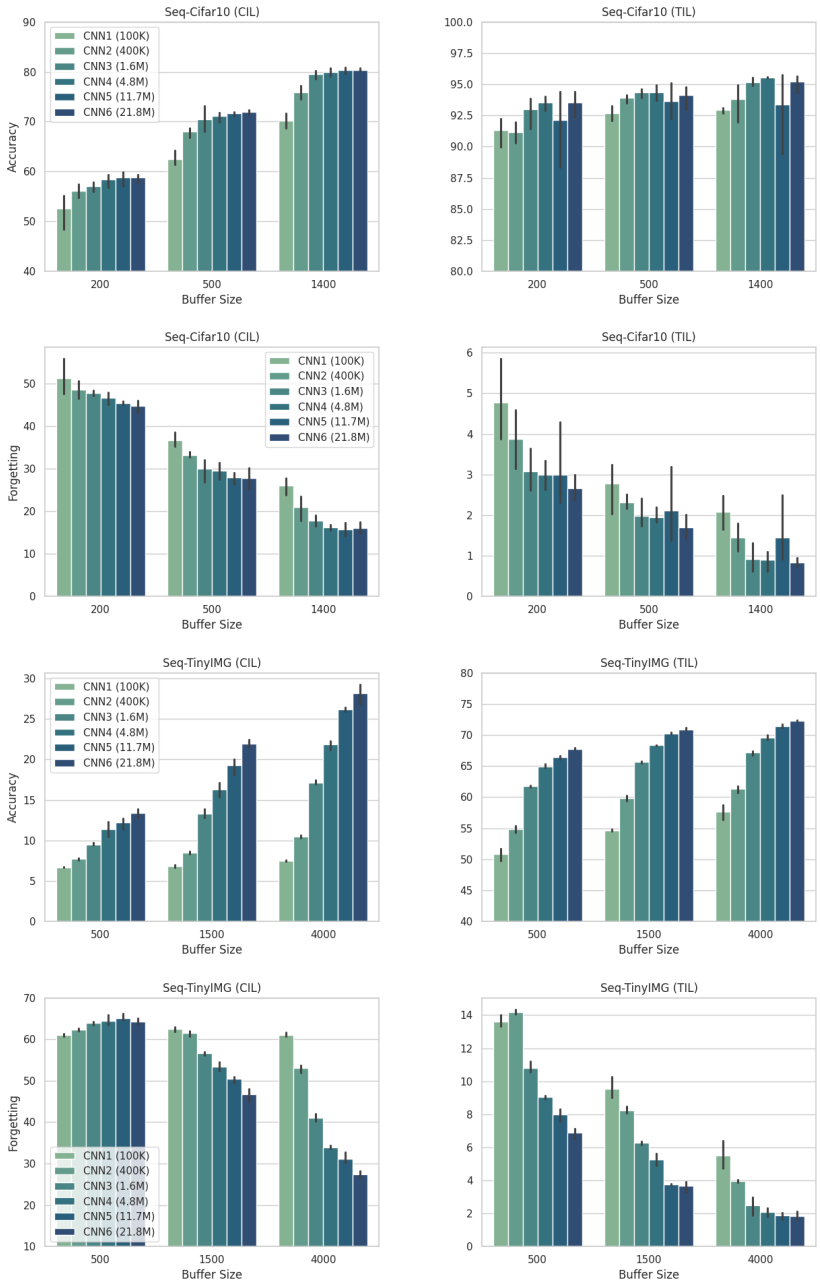

Figure 13: **Scaling CNN improves the performance.** Classification results for DER offline on standard CL benchmarks when scaling a vanilla CNN. Results are averaged across 3 different seeds.