SCALE: Online Self-Supervised Lifelong Learning without Prior Knowledge Supplementary Material

In the supplementary, we include more details on the following aspects:

- In Section 1, we list the **implementation details of SCALE, lifelong learning baselines and self-supervised learning baselines**, especially the hyperparameters for each dataset. For SCALE, we detail the online memory update algorithm and compare with MinRed [15].
- In Section 2, we provide details on **constructing data streams** in our online unsupervised lifelong learning problem setup.
- In Section 3, we show the **accuracy curves** during training on all datasets. The accuracy curves of all lifelong learning baselines and SCALE are complementary to the results in Section 6 of the main paper.
- In Section 4, we conduct sensitivity analyses on the **streaming batch size** n **and memory batch size** m in SCALE.
- In Section 5, we conduct sensitivity analyses on the **temperature** τ in our pseudo-supervised contrastive loss. Different temperatures are ideal for *iid* and non*iid* streams.
- In Section 6, we present the **t-SNE plots** of the features during periodic evaluation, which vividly demonstrates SCALE's learning process.
- In Section 7, we analyze the **computation time complexity** of SCALE, including all components of pseudo-contrastive loss, forgetting loss and memory update.

1. Implementation Details

1.1. Implementation Details of SCALE

We implement the pseudo-supervised contrastive learning component of SCALE based on the official SupCon framework [10]. We use ResNet-18 [8] with a feature space dimension of 512 as backbone. We use the Stochastic Gradient Descent (SGD) optimizer with learning rate of 0.03. The hyperparameters across all datasets are summarized in Table 1.

Table 1. Hyperparameters of SCALE across all datasets.

Param.	Explain.	Value
lr	Learning rate	0.01
n	Batch size for streaming data	128
M	Memory buffer size	1280
m	Sampled memory batch size	128
τ	Temperature for pseudo-contrastive loss	0.1
μ	Relative similarity threshold	0.05
λ	Weight for self-supervised forgetting loss	0.1

Data augmentation. All methods except STAM share the same augmentation procedure. For STAM, we use their official data loader with custom pre-processing. During the training phase, our data augmentation procedure first normalizes the data using mean and variances. we apply random scaling 0.2-1, random horizontal flip, random color jitter of brightness 0.6-1.4, contrast 0.6-1.4, saturation 0.6-1.4, hue 0.9-1.1, and random gray scale with $p = 0.2$ for CIFAR-10 and CIFAR-100. For TinyImageNet, we apply the random scaling 0.08-1 with random aspect ratio 0.75-1.33 and bicubic interpolation. All images are resized to 32 $\times$ 32. During the evaluation phase, we only normalize the data but do not use any augmentation for all datasets.

Uniform memory subset sampling. In SCALE, one key component is the online memory update where we adapt the uniform subset sampling algorithms. To achieve the best performance in online unsupervised lifelong learning (ULL), the memory buffer is supposed to retain the most "representative" samples regarding the historical distribution in the feature space. Uniform sampling mechanism is desired to extract representative samples from the sequential imbalanced streams. To remind the readers, the input to the memory update is the imbalanced combined features $\{\mathbf{z}_i\}_{i=1}^{M+n}$ projected by the latest model θ_u^t from the previous raw memory samples $\{\mathbf{e}_i\}_{i=1}^{M}$ and latest streaming batch X_u^t. The goal is to select a subset of M samples from $\{\mathbf{z}_i\}_{i=1}^{M+n}$ then store the corresponding raw samples in the new memory buffer. We employ the Part and Selection Algorithm (PSA) [18] in SCALE and adapt the implementation from `diversipy` (`https://github.com/DavidWalz/diversipy`). The implementation is the slightly improved version from [20]. PSA is a linear-time algorithm designed to select a subset of well-spread points. The algorithm has two stages: first, the candidate

set $\{\mathbf{z}_i\}_{i=1}^{M+n}$ is partitioned into M subsets, then one member from each subset is selected to form the updated memory. During the first stage, each partition step selects the set with the greatest dissimilarity among its members to divide. The dissimilarity of a set $A = \{\mathbf{z}_i\}_{i=1}^{M+n}$ is defined as the maximum absolute difference among all dimensions:

$$a_j := \min_{i=1,...,M+n} z_{ij}, \quad b_j := \max_{i=1,...,M+N} z_{ij},$$
$$\Delta_j = b_j - a_j, \quad j = 1, ..., K \tag{1a}$$
$$\emptyset A := \max_{j=1,...,K} \Delta_j \tag{1b}$$

where K denotes the dimension of the feature space $\mathcal{Z}$. The dissimilarity of A is the diameter of A in the Chebyshev metric. During the second stage, PSA chooses the closest member (in Euclidean metric) to the center of the hyperrectangle around A_i. The pseudocode and complexity analysis of PSA are presented in [18] and [20]. The execution time of PSA in our setup is discussed in Section 7.

Comparison of MinRed and PSA (in SCALE). The latest study by Purushwalkam *et al.* [15] proposed a minimum redundancy (MinRed) memory update policy, which assists buffer replay in self-supervised learning. When the number of samples in the memory exceeds its capacity, they rely on the cosine distance between all pairs of samples to discard the most redundant one:

$$i^* = \arg\min_i \min_{j \neq i} d_{cos}(\mathbf{z}_i, \mathbf{z}_j) \tag{2}$$

Intuitively, MinRed is a greedy heuristic that keeps the most "disimilar" M samples. The "dissimilarity" is judged by the greatest distance from its closest selected feature. Although MinRed is effective in retaining diverse samples, it does not take into account global distribution and may lead to biased selection on imbalanced incoming streams. This leads to a degraded performance, as shown in Table 3 of the main paper.

1.2. Implementation Details of Lifelong Learning Baselines

The following lifelong learning baselines are used to compare with SCALE:

- **PNN** [17]: Progressive Neural Network gradually expands the network architecture.
- **SI** [22]: Synaptic Intelligence performs online per-synapse consolidation as a typical regularization technique.
- **DER** [1]: Dark Experience Replay retains existing knowledge by matching the network logits across a sequence of tasks.
- **STAM** [19] uses online clustering and novelty detection to update an expandable memory architecture.
- **CaSSLe** [6] proposes a general framework that extracts the best possible representations invariant to task shifts in ULL.
- **LUMP** [12] interpolates the current with the previous samples to alleviate catastrophic forgetting in ULL. Use SimCLR.

Note, that all methods except STAM are addable to self-supervised learning backbones, while STAM employs a unique expandable memory architecture. As SCALE lies on the SimCLR backbone, we also experiment with the above baselines on the SimCLR backbone for a fair comparison. We did not compare with VAE-based methods such as [9, 16] since they have been reported to scale poorly on large image datasets [5]. More implementation details are grouped and summarized as follows:

- **PNN**, **SI**, **DER**, **LUMP** are adapted from the official framework in [12] using their default hyperparameters. PNN, SI and DER are originally designed for supervised lifelong learning but are adapted to ULL tasks as described in [12]. For fair comparison, we use SimCLR as the underlying contrastive learning backbone for these baselines. For DER and LUMP, we use a replay buffer of the same size as SCALE.
- We take advantage of the official implementation of **STAM** on CIFAR-10 and CIFAR-100 with their default hyperparameters. We use the original data loader and parameters for CIFAR-10, CIFAR-100 as in the released code, and use our clustering and kNN classifier on the learned embeddings.
- We use a modified version **CaSSLe** based on the original implementation. Specifically, we remove task labels and force the model to compare the representations of the current and previous batch.

2. Data Streams Construction

To remind the reader, we evaluated three image datasets: CIFAR-10 (10 classes) [13], CIFAR-100 (20 coarse classes) [11] and a subset of ImageNet (10 classes) [4]. We construct five single-pass data streams for training:

- ***iid* stream:** We sample 4096, 2560 and 500 images from each class of CIFAR-10, CIFAR-100, and TinyImageNet, then shuffle all samples.
- **Sequential class-incremental stream:** We sample 4096, 2560 and 500 images from each class of CIFAR-10, CIFAR-100, and TinyImageNet, then feed them class-by-class to the model.
- **Sequential class-incremental stream with blurred boundaries:** We sample the same number of images from each class as the standard sequential class-incremental stream. We then mix the last 25% samples of the previous class with the first 25% samples of the

next class, with a gradual mix probability between 0.05 and 0.5. Specifically, for samples closer to the boundary, there is a higher probability to be exchanged with a sample on the other side of the boundary.

- **Sequential class-incremental stream with imbalanced class appearance:** For each incrementally introduced class, we randomly sample a subset with more than half of the total samples in that class. Specifically, suppose that there are U samples in that class. We first uniformly sample an integer $V \in [0.5U, U]$, then we randomly sample V samples from that class.
- **Sequential class-incremental stream with concurrent class appearance:** Similar as the sequential class-incremental stream, we sample the same amount of images from each class. We then group the classes 2-by-2 with its adjacent class, and shuffle all samples in one group. In this way, each 2-class group is revealed to the model incrementally, while the samples in one group follow a random order.

For the evaluation dataset, we sample 500, 250 and 50 samples per class from the official validation dataset of CIFAR-10, CIFAR-100 and TinyImageNet respectively.

3. Accuracy Curve during Training

The accuracy curves of all lifelong learning methods during training are depicted in Figure 6, 7 and 8 for CIFAR-10, CIFAR-100 and TinyImageNet respectively. Outstanding from all methods, SCALE learns incrementally regardless of the *iid* or sequential manner. Compared to *iid* cases, sequential data streams are more challenging, where more baselines present the "forgetting" or unimproved trend as new classes arrive. Among the three datasets, CIFAR-10 streams are easier to learn from. CIFAR-100 streams with 20 coarse classes act as the most challenging dataset where multiple baselines collapse from the beginning. The 10-class subset from ImageNet causes more fluctuations during the online learning procedure.

4. Sensitivity Analyses of Streaming and Memory Batch Sizes

As indicated in multiple studies [3,7,21], batch size has a significant impact on the performance of contrastive learning methods, as a large number of samples are required to enhance the contrast effect. We study the impact of streaming and memory batch sizes in SCALE. We first fix the memory batch size $m = 128$ and alter the streaming batch size upon *iid* and sequential CIFAR-10 streams. The average final ACC and kNN accuracy after 3 random trials are shown in Figure 1. It can be seen that the impact of batch sizes on ACC and kNN accuracies is slightly different. Compared to ACC, kNN accuracy behaves more stably

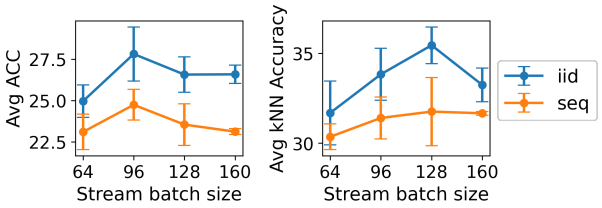


Figure 1. Average ACC (left) and kNN accuracy (right) on *iid* and sequential CIFAR-10 streams, with different batch sizes n and memory batch size $m = 128$.

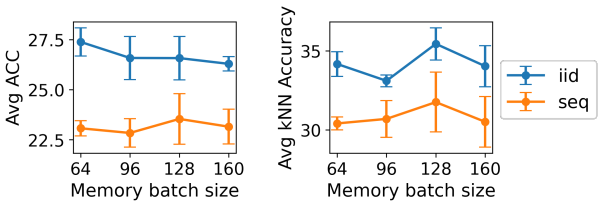


Figure 2. Average ACC (left) and kNN accuracy (right) on *iid* and sequential CIFAR-10 streams, with different memory batch sizes m and streaming batch size $n = 128$.

hence our discussion in the rest of the material mainly focuses on kNN accuracies. For *iid* streams, a larger batch size leads to a higher kNN accuracy in SCALE, as more samples can be used for contrast. However, in the sequential case, SCALE is robust to batch sizes with less than 1% difference in terms of kNN accuracy when using batch sizes of 64, 96, 128 and 160. Such robustness can be attributed to two reasons: (i) unlike SimCLR, we use small batch sizes for the online learning scenarios, thus the effect of varying batch sizes diminishes; (ii) for the sequential streams, the contrasting samples mainly come from the memory buffer (with different labels). Therefore a large batch size does not greatly improve the contrastive learning performance.

We then fix the streaming batch size to $n = 128$ and apply various memory batch sizes. The average ACC and kNN accuracy of SCALE on 3 random CIFAR-10 streams is shown in Figure 2. Interestingly, as the contrasting performance of SCALE depends on both the streaming and memory samples, the effect of changing one of them is not significant. When using memory samples of 64, 96, 128 and 160 on sequential streams, the different on ACC and kNN accuracies are less than 0.7% and 1.35% respectively.

5. Sensitivity Analyses of Temperature τ

We setup MNIST following similar protocols in Section 2. Figure 3 reports the ACC at the end of *iid* and single-class sequential data streams on MNIST, when choosing various values for temperature τ in the contrastive loss (Equation (3)) and temperature κ in the tSNE pseudo-positive set selection (Equation (6)). It can be observed that

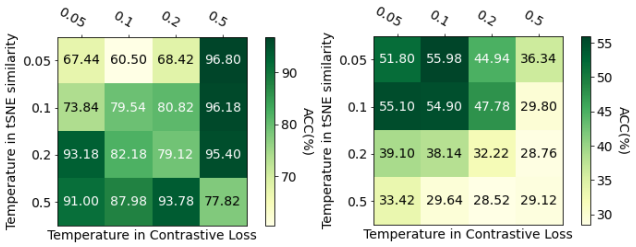


Figure 3. Heatmap of final ACC on MNIST, *iid* stream (left) and sequential class-incremental stream (right) using various temperatures.

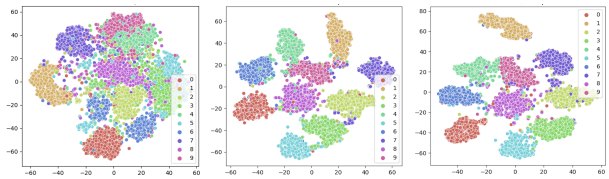


Figure 4. t-SNE plots on the evaluation dataset at the start (left), middle (middle) and end (right) of training on *iid* data streams on MNIST.

the type of data stream (i.e., *iid* or sequential) has a significant effect on the best combinations of temperatures. Under the *iid* datastream, high temperature of $\tau = 0.5$ is preferred while κ has a small impact on the final ACC. However, in the sequential case, temperature of $\tau = 0.1$ or even smaller is desired while κ in pseudo-positive set construction also drives the final ACC. Intuitively, contrastive learning benefits when there are more negative samples from the other classes to compare against, where a large temperature value works better. However, in online ULL scenarios, a lower temperature τ with comparable κ shows better performance in driving the closer samples together and memorizing the similarity relationship.

6. t-SNE Plots during Training

To clearly visualize the challenges of learning from sequential incremental input versus *iid* input, we depict the t-SNE plots on the feature space using the evaluation dataset during training SCALE. The colors indicate ground-truth class labels. As shown in Figure 4, under *iid* data streams on MNIST, all classes are quickly separated as the middle-stage t-SNE plot already demonstrates the distinguished class distribution in the feature space. On the contrary, due to the lack of labels and balanced data input, distinguishing and memorizing various classes under class-incremental input is much more difficult as shown in Figure 5. SCALE is able to extract obvious class patterns and discriminate one class versus the others by the kNN classifier.

7. Time Complexity of SCALE

Time complexity of loss functions. We analyze the computation complexity of SCALE and compare with state-of-the-art lifelong learning method.

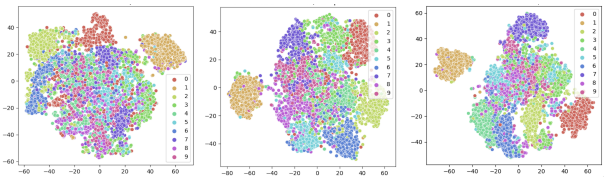


Figure 5. t-SNE plots on the evaluation dataset at the start (left), middle (middle) and end (right) of training on sequential data streams on MNIST.

- **Co2L** [2] is the state-of-the-art supervised lifelong learning method using contrastive loss and forgetting loss. Both losses depend on the pairwise similarity between all streaming and memory representations. Hence after the forward propagation, the computation complexity of computing the losses is $O((m+n)^2)$, where m and n refer to the memory and streaming batch size respectively.
- **SCALE** utilizes the pseudo-contrastive loss and forgetting loss, both based on pairwise similarity and the computation can be reused. Therefore, the computation complexity to compute the losses in SCALE is the same as Co2L, both being $O((m+n)^2)$. Moreover, SCALE consumes less time and resource than CaSSLe without the predictor.

We measure the execution time per batch on a Linux desktop with Intel Core i7-8700 CPU at 3.2 GHz and 16 GB RAM, and a NVIDIA GeForce 3080Ti GPU. The settings are the same as the implementation details in Section 1. The results in Table 2 show that SCALE consumes nearly the same time as Co2L. The computation time of Co2L and SCALE is directly affected by the combined batch size $m+n$, which supports our analyses.

Table 2. Average computation time (in seconds) of losses per batch in Co2L, CaSSLe and SCALE on CIFAR-10, using various batch sizes.

n	m	Time (s)	
		Co2L	SCALE
128	128	0.050	0.051
64	128	0.034	0.035
128	64	0.034	0.035

Time complexity of memory update. SCALE employs the PSA to select a uniformly distributed subset. We measure the execution time per memory update of random selection, KMeans-based selection, MinRed [15] and PSA on the same machine. The results are summarized in Table 3. The configurations are the same as described in Section 1. PSA is only slower than the random baseline and executes faster than KMeans-based selection and MinRed. The KMeans-based selection performs KMeans clustering on all

latent features and then runs a random update within each cluster. We implement KMeans using the `scikit-learn` library [14] with k equal to the ground-truth number of classes. In our setting, KMeans is not ideal as it not only uses prior knowledge of the number of class, but is not computationally efficient due to its iterative nature. MinRed as a greedy heuristic needs to evaluate all candidates in a sample-by-sample manner. In our implementation, MinRed is 10% slower than PSA.

While the memory update seems to take much longer time compared to computing loss values, we remind the reader that all above memory selection mechanisms are deployed on CPU, and do not utilize the acceleration capability of GPU. In the future, we plan to re-implement the code to convert to a GPU version.

Table 3. Average computation time (in seconds) per memory update on CIFAR-10, using various memory update policies.

	random	KMeans	MinRed [15]	PSA
Time (s)	0.40	1.51	1.05	0.95

References

[1] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

[2] Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co2l: Contrastive continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9516–9525, 2021.

[3] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[5] William Falcon, Ananya Harsh Jha, Teddy Koker, and Kyunghyun Cho. Aavae: Augmentation-augmented variational autoencoders. *arXiv preprint arXiv:2107.12329*, 2021.

[6] Enrico Fini, Victor G Turrisi da Costa, Xavier Alameda-Pineda, Elisa Ricci, Karteek Alahari, and Julien Mairal. Self-supervised models are continual learners. *arXiv preprint arXiv:2112.04215*, 2021.

[7] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 1965–1972, 2017.

[10] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *arXiv preprint arXiv:2004.11362*, 2020.

[11] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[12] Divyam Madaan, Jaehong Yoon, Yuanchun Li, Yunxin Liu, and Sung Ju Hwang. Representational continuity for unsupervised continual learning. In *International Conference on Learning Representations*, 2022.

[13] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Senthil Purushwalkam, Pedro Morgado, and Abhinav Gupta. The challenges of continuous self-supervised learning. pages 702–721, 2022.

[16] Dushyant Rao, Francesco Visin, Andrei A Rusu, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Continual unsupervised representation learning. *arXiv preprint arXiv:1910.14481*, 2019.

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

[18] Shaul Salomon, Gideon Avigad, Alex Goldvard, and Oliver Schütze. Psa–a new scalable space partition based selection algorithm for moeas. In *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, pages 137–151. Springer, 2013.

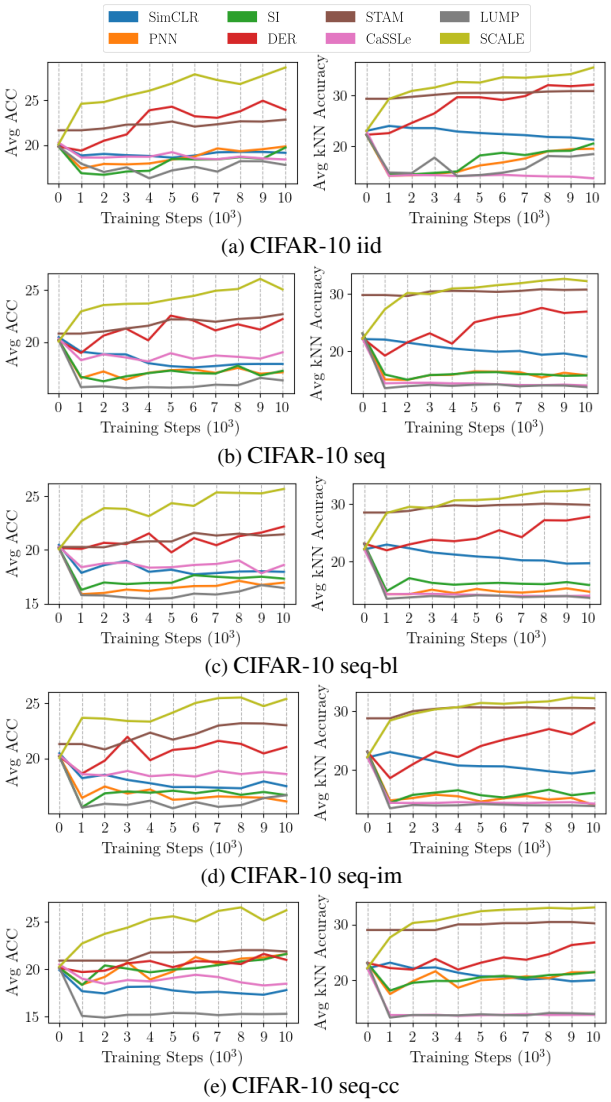
[19] James Smith, Cameron Taylor, Seth Baer, and Constantine Dovrolis. Unsupervised progressive learning and the stam architecture. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2979–2987. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[20] Simon Wessing. *Two-stage methods for multimodal optimization*. PhD thesis, Dissertation, Dortmund, Technische Universität, 2015, 2015.

[21] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320. PMLR, 2021.

[22] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International*

Conference on Machine Learning, pages 3987–3995. PMLR, 2017.



(a) CIFAR-10 iid

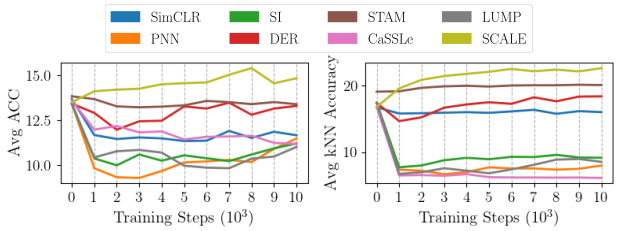
(b) CIFAR-10 seq

(c) CIFAR-10 seq-bl

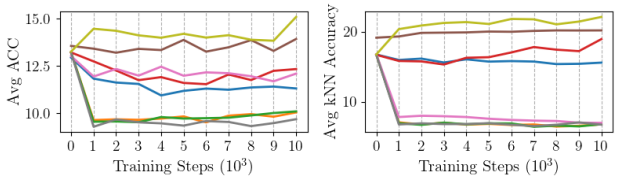
(d) CIFAR-10 seq-im

(e) CIFAR-10 seq-cc

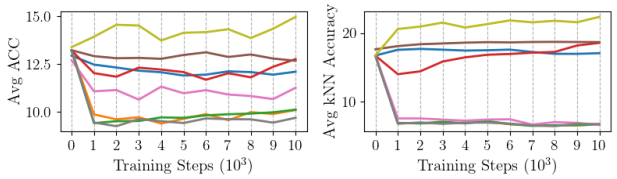
Figure 6. ACC and kNN accuracy curve on all streams sampled from CIFAR-10 using various lifelong learning baselines.



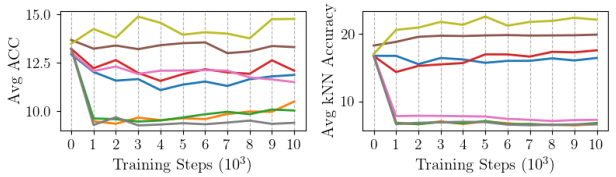
(a) CIFAR-100 iid



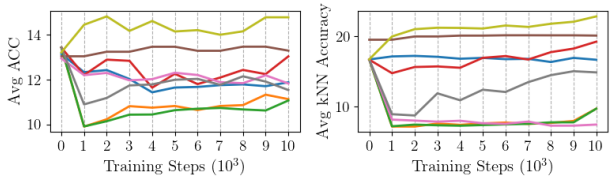
(b) CIFAR-100 seq



(c) CIFAR-100 seq-bl

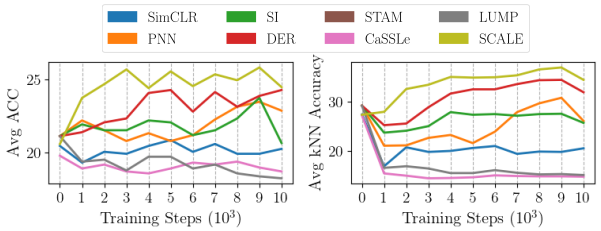


(d) CIFAR-100 seq-im

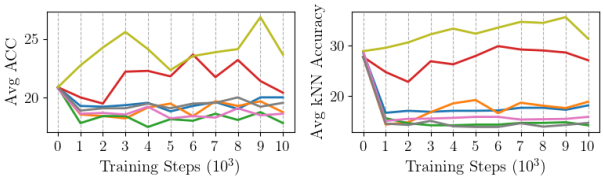


(e) CIFAR-100 seq-cc

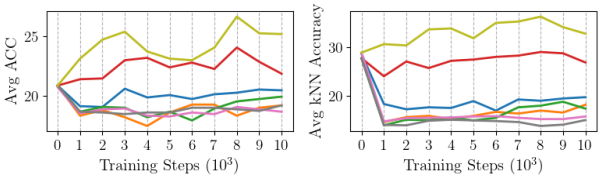
Figure 7. ACC and kNN accuracy curve on all streams sampled from CIFAR-100 using various lifelong learning baselines.



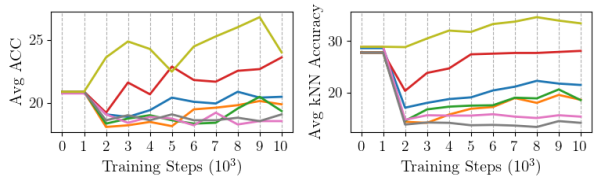
(a) TinyImageNet iid



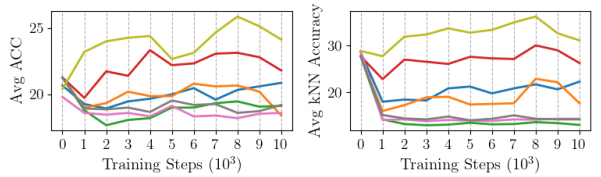
(b) TinyImageNet seq



(c) TinyImageNet seq-bl



(d) TinyImageNet seq-im



(e) TinyImageNet seq-cc

Figure 8. ACC and kNN accuracy curve on all streams sampled from TinyImageNet using various lifelong learning baselines.