# Speeding Up Online Self-Supervised Learning by Exploiting Its Limitations

Sina Mokhtarzadeh Azar and Radu Timofte

Computer Vision Lab, CAIDAS & IFI, University of Würzburg, Germany
`{sina.mokhtarzadeh-azar, radu.timofte}@uni-wuerzburg.de`

**Abstract.** Online Self-Supervised Learning tackles the problem of learning Self-Supervised representations from a stream of data in an online fashion. This is the more realistic scenario of Self-Supervised Learning where data becomes available continuously and must be used for training straight away. Contrary to regular Self-Supervised Learning methods where they need to go hundreds of times through a dataset to produce suitable representations, Online Self-Supervised Learning has a limited budget for training iterations for the new data points from the stream. Additionally, the training can potentially continue indefinitely without a specific end. We propose a framework for Online Self-supervised Learning with the goal of learning as much as possible from the newly arrived batch of data in a limited amount of training iterations before the next batch becomes available. To achieve this goal we use a cycle of aggressive learning rate increase for every batch of data which is combined with a memory to reduce overfitting on the current batch and forgetting the knowledge gained from previous batches. Additionally, we propose Reducible Anchor Loss Selection (RALS) to intelligently select the most useful samples from the combination of the new batch and samples from the memory. Considering the limitation of a smaller number of iterations over the data, multiple empirical results on CIFAR-100 and ImageNet-100 datasets show the effectiveness of our approach.

**Keywords:** Self-Supervised Learning · Continual Learning.

## 1 Introduction

Large amounts of unlabeled visual data are being generated every day from multiple sources like smartphones, surveillance cameras, self-driving cars, etc. Self-supervised Learning (SSL) can make use of this limitless data to learn strong representations that can be used to improve various downstream tasks. However, training on the possibly infinite data is a real challenge. Recent methods like [10,17] provide models that are able to learn continually from unlabeled data. These methods process large chunks of data in the form of new tasks on which they usually spend a significant amount of training epochs. In a scenario where new data arrives in an online fashion from a stream, these approaches are not usable due to the long training process. Therefore, Online Self-Supervised Learning (OSSL) methods capable of learning from streaming data must be developed.

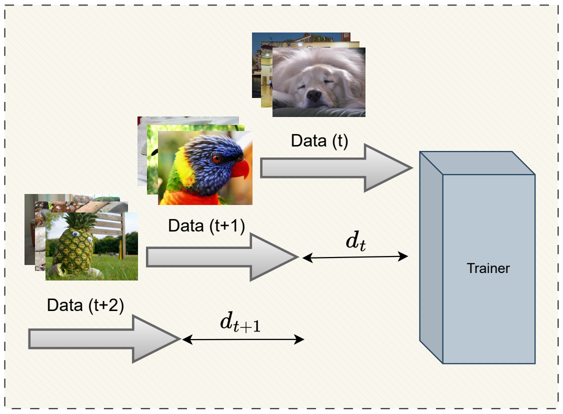


Fig. 1: Illustration of a scenario where a stream is generating batches of data with a time difference between each batch. Trainer has a limited amount of time to learn from the currently available batch of data before the next batch arrives.

Considering a stream of data where there is limited time between the preparation of each data batch from the stream (see Figure 1), we need to restrict the number of training iterations on the batch in order for the training to continue on the new batch. With a potentially infinite data stream, the end of training is unknown. Therefore, another challenge is that the usual learning rate schedulers requiring the end of training to be specified can not be used out of the box in the online learning setting.

We propose Single Local Cycle Learning Rate Scheduler (SLCLRS) in order to tackle the challenges of infinite data with an unknown ending and the need for faster learning from the currently available batch of data with limited training steps. This is inspired by [28] which uses the One Cycle Learning Rate Scheduler (OCLRS) to increase the learning rate to a large value midway through the training epochs and then gradually decrease it to a very small value towards the ending epoch which in turn speeds up the training process. However, this scheduler works based on the whole training process. In order to tackle the online scenario we use SLCLRS with a limited number of training steps for every new batch that arrives to address the previous challenges.

Additionally, we propose Reducible Anchor Loss Selection (RALS) to select the most suitable samples in the online learning setting considering the change of the loss for the samples based on the current model and the anchor model trained prior to getting the current batch of data. This is based on Reducible Holdout Loss Selection (RHO-LOSS) [19] that follows the direction of choosing the most suitable samples for training based on a loss increase from a pre-trained model on a hold-out set. However, RHO-LOSS works in an offline setting with a static model. RALS on the other hand uses a dynamic anchor model to calculate the selection criteria.

The rest of the paper is organized as follows. First, in Sec. 2, the related works are briefly discussed. A formalized problem definition is provided in Sec. 3. Details of our framework are given in Sec. 4. This is followed by experiments and analysis in Sec. 5. We conclude the paper in Sec. 6.

## 2 Related Works

**Continual Learning.** Continual Learning has the goal of learning new tasks continually without forgetting the previously learned tasks. The main techniques used to tackle the Continual Learning problem are Regularization-based, Memory-based, or Parameter Isolation-based.

Regularization-based methods simply add a regularization term to limit the changes happening to the parameters of the model to stop it from forgetting the previous tasks [15,1,3,32,23,16,6,12].

Memory-based methods keep a portion of data in a memory [2,4,5,22,24]. By combining the samples in the memory with the samples in the new task, catastrophic forgetting can be mitigated.

Among memory-based methods, Maximally Interfered Retrieval (MIR) [2] is the most similar to our work. They use a combination of samples from memory and the new task to train a supervised model. An optimization step is performed using only the new samples to find the samples in the memory with the highest increase in loss to be retrieved for training.

Parameter Isolation based methods tackle forgetting by using different parameters for each task [18,9,27,26,30]. This can be achieved by using different parameters inside a fixed model for each task or by adding new parameters for every new task.

**Continual Self-Supervised Learning.** Continual Self-Supervised Learning is a relatively new topic. Lifelong Unsupervised Mixup (LUMP) [17] shows that features learned using unsupervised continual learning provide better generalization and suffer less from catastrophic forgetting. They also propose to interpolate samples from a memory with the current task samples to reduce forgetting.

CaSSLe [10] proposes a Continual Self-Supervised Learning approach that utilizes a distillation mechanism compatible with various self-supervised Learning algorithms instead of a memory to reduce forgetting and it achieves impressive results. However, the training process takes a lot of epochs which makes it unable to work in an online setting.

Most recently, in the concurrent work of [21] the problem of Continuous Self-Supervised Learning is studied. They mainly address the Non-IID nature of correlated streaming data through the use of a Minimum Redundancy Buffer. Our method is similar to this work with the difference that we focus on speeding up the self-supervised Learning convergence on the provided data stream without explicitly handling Non-IID data. Similar ideas like [21] can be used in combination with our approach to implicitly consider the Non-IID data but it is not the focus of this paper.

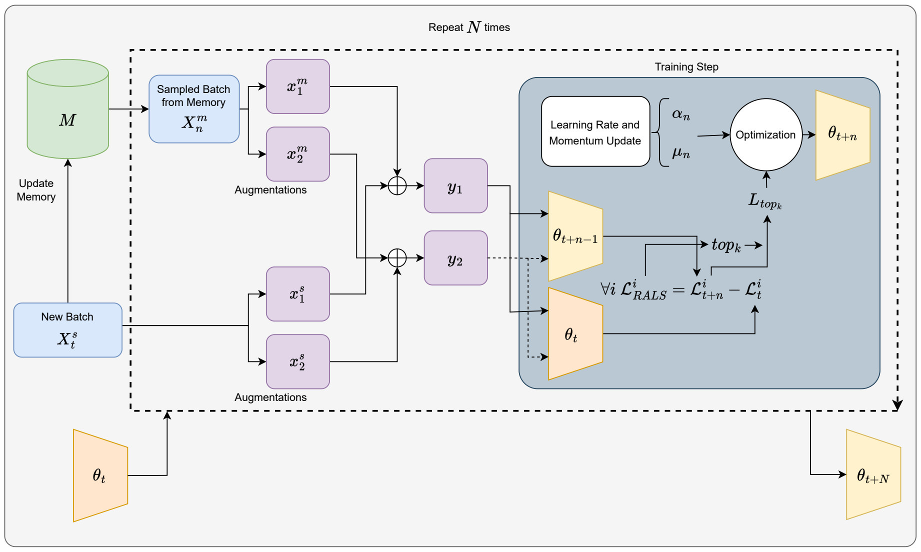


Fig. 2: Overview of our proposed method. Every batch of data $X_t^s$ that arrives at time $t$ from the stream is used in combination with a batch of data $X_n^m$ from a memory $M$ for training a model for a limited number of steps $N$ using the single local cycle learning rate scheduler before the next batch gets ready. Reducible anchor loss selection is also used to find the best possible subset of samples in order to provide even more speed up for training.

## 3 Problem Definition

Regular Self-Supervised Learning methods have access to all the samples in the dataset $D$ during training. In each training step, a batch of images $X$ is sampled from $D$ and two augmentations $x_1$ and $x_2$ are generated from it. These augmentations are fed to a model $f$ to produce representations $z_1$ and $z_2$. $f$ is parameterized by $\theta = (\theta_b, \theta_p)$ where $\theta_b$ and $\theta_p$ are the parameters for backbone and projection layers, respectively. A self-supervised loss function $\mathcal{L}_S(z_1, z_2)$ is used to optimize the parameters $\theta$ based on $z_1$ and $z_2$:

$$\sum_b^B \mathcal{L}_S(z_1^b, z_2^b). \tag{1}$$

In Continual Self-supervised Learning (CSSL) the model is trained on a sequence of $T$ tasks on different corresponding chunks of the dataset $D = (D_1, D_2, ..., D_T)$. These chunks of data are sequentially provided to the model without access to the other chunks. However, it considers no limits on the time and epochs to train the model on every subset of $D$. The model must be trained in a way that in addition to learning from new data chunks, it does not forget what was learned from previous chunks. Therefore self-supervised loss on all the subsets of $D$ must be minimized:

$$\sum_{t}^{T}\sum_{b}^{B_t}\mathcal{L}_S(z_1^{b,t}, z_2^{b,t}). \tag{2}$$

We address the more realistic setting where data source $D$ is a stream (with possibly non i.i.d data) and the training must happen in an online approach. Every batch of data $X_t^s$ that arrives at time $t$ from the stream must be used for training for a limited amount of time before the next batch gets ready. The objective for Online Self-Supervised Learning (OSSL) can be considered as an extreme case of CSSL where the chunks of data are small and the number of OSSL tasks $T_O$ is much larger than CSSL ones ($T_O \gg T$). Therefore, the objective will be similar to CSSL:

$$\sum_{t}^{T_O}\sum_{b}^{B_t}\mathcal{L}_S(z_1^{b,t}, z_2^{b,t}). \tag{3}$$

Our goal is to learn representations using $X_t^s$ as fast as possible without having access to future data or the past data except the ones stored in a memory $M$. Additionally, we avoid using learning rate schedulers that consider an end for the data stream.

## 4 Proposed Method

The goal of our approach is to improve the speed of convergence for Online Self-Supervised Learning considering the challenges of the unknown end of training and a limited number of training iterations per new batch of data from the stream. In this section, we describe the proposed framework and describe how SLCLRS and RALS are incorporated into it. An overview of our approach is presented in Figure 2.

### 4.1 Single Local Cycle Learning Rate Scheduler

In a regular training setting OCLRS is used to achieve comparable performance with fewer training epochs by increasing the learning rate to a high value at the beginning of the training and reducing it to a much smaller value at the end. We suggest that a batch of data is the small version of the dataset and we can use a single cycle of increasing and decreasing the learning rate in the current local iteration corresponding to the new batch of data $X_t^s$ in order to speed up learning this batch. The duration of this cycle is based on the available budget for the training steps in the current local iteration.

Using an aggressive scheduler on just the new batch $X_t^s$ will lead to overfitting on that batch and possibly forgetting the previously learned representations. Therefore, we store a portion of data in the memory $M$ and combine a batch from memory $X_n^m$ with the new batch from stream $X_t^s$ in every training step $n$ (similar to [2]). Given the optimizer $o$ and scheduler $s$ for $n \in 1, ..., N$ where $N$

is the number of training steps for each batch of data, the training process can be written as follows:

$$\begin{aligned} y_1 &= x_1^m \oplus x_1^s, \\ y_2 &= x_2^m \oplus x_2^s, \end{aligned} \tag{4}$$

$$\begin{aligned} z_1^{n-1} &= f_{\theta_{t+n-1}}(y_1), \\ z_2^{n-1} &= f_{\theta_{t+n-1}}(y_2), \end{aligned} \tag{5}$$

$$\mathcal{L}_{t+n} = \mathcal{L}_S(z_1^{n-1}, z_2^{n-1}), \tag{6}$$

$$\begin{aligned} \alpha_n, \mu_n &= s(n), \\ \theta_{t+n} &= o(\mathcal{L}_{t+n}, \theta_{t+n-1}, \alpha_n, \mu_n). \end{aligned} \tag{7}$$

Here, $(x_1^m, x_2^m)$ and $(x_1^s, x_2^s)$ are the augmentations created from $X_n^m$ and $X_t^s$, respectively. $\oplus$ shows the concatenation along the batch dimension and $(y_1, y_2)$ are the concatenated augmentations. $f_{\theta_{t+n-1}}$ is the model optimized for $n-1$ steps after receiving the batch of data at step $t$. This model is used to produce the representations $(z_1^{n-1}, z_2^{n-1})$ which in turn are used to calculate the loss $\mathcal{L}_{t+n}$ at step $n$. The scheduler updates the learning rate $\alpha_n$ and momentum $\mu_n$ based on step $n$ then the optimizer calculates the new set of parameters $\theta_{t+n}$ for step $n$. The reason we also update the momentum is that the positive effect of reducing it while increasing the learning rate was observed in [28].

The scheduler in [28] uses linear functions for increasing and decreasing the learning rate. However, we utilize the cosine function version provided in PyTorch [20] which is in turn based on Fastai's implementation [13]. An illustration of the learning rate and momentum schedules we use with a different number of training steps per batch is provided in the Appendix.

As previously mentioned, one of the challenges of OSSL is the possibly unknown ending. The benefit of using SLCLRS with every new batch is that we do not need to consider an ending for the training and this procedure can continue indefinitely given a data stream. The training can stop at the end of every iteration knowing sufficient optimization was performed on the latest data from the stream.

It should be noted that we use simple random sampling for both memory update and retrieval. More complex strategies to handle the memory can be used for more efficient memory management but it was not the focus of our work.

### 4.2 Reducible Anchor Loss Selection

The use of memory in combination with different augmentations $(x_1^s, x_2^s)$ at each training step $n$ can reduce the chances of overfitting on the new batch $X_t^s$ but it can still happen. Additionally, we suggest a scenario exists that the loss for some samples of $X_t^s$ stops decreasing which in turn slows down the convergence on the combined set of samples. Therefore, selecting samples with the highest increase in loss compared to the beginning step of this iteration can help speed up the convergence. The inspiration for this idea comes from the reducible holdout loss selection of Mindermann *et al* [19]. Given a model for training, this method finds the data points with the highest increase in loss compared to the loss of the pre-trained model on a holdout set. These points are then used for training. However, in our use case, we do not necessarily have a holdout set. Instead, we consider the model at the beginning of the iteration $f_{\theta_t}$ as an anchor model to also limit the divergence of parameters from the anchor point. Our method with RALS can be described as follows:

$$\begin{aligned} z_1^0 &= f_{\theta_t}(y_1), \\ z_2^0 &= f_{\theta_t}(y_2), \end{aligned} \tag{8}$$

$$\begin{aligned} \mathcal{L}_t^i &= \mathcal{L}_S^i(z_1^0, z_2^0), \\ \mathcal{L}_{RALS}^i &= \mathcal{L}_{t+n}^i - \mathcal{L}_t^i, \\ \mathcal{L}_{top_k} &= \frac{1}{k} \sum_{j \in top_k(\mathcal{L}_{RALS}^i)} \mathcal{L}_{t+n}^j \end{aligned} \tag{9}$$

Here, $\mathcal{L}_{t+n}^i$ shows the loss at step $n$ for the individual sample $i$. Reducible anchor loss $\mathcal{L}_{RALS}^i$ is the increase in loss for sample $i$ and it is used as the metric for the selection process. Finally, based on the reducible anchor loss metric, the average loss over top $k$ samples is calculated as $\mathcal{L}_{top_k}$, and the optimization is performed based on this loss. This approach should lead to faster convergence by focusing on the more important points. In practice in order to find the top $k$ samples, initially we run the models in inference mode without calculating gradients. Then only the selected samples are fed again to the model for training.

Multiple scenarios can lead to a data sample having high reducible anchor loss. First, if the model started overfitting on a sample at step $n$, its loss value for that sample will start to decrease compared to the frozen anchor model. Therefore, due to lower $\mathcal{L}_{RALS}^i$, the probability of being selected among the top RALS samples would decrease leading to less overfitting on that sample. Second, a sample from the memory would probably have a lower loss with $f_{\theta_t}$. If this loss increases at step $n$, it could mean that the model has started to forget the representations learned from this sample. Therefore, it will be selected based on reducible anchor loss for training which in turn will lead to less forgetting. Finally, similar to [19], less relevant and noisy samples will also not get into selected samples due to already high $\mathcal{L}_t^i$.

## 4.3 Integration of SSL Methods

We studied two popular Self-Supervised Learning methods of SimCLR [7], and Barlow Twins [31] in our proposed framework. A summary of each method and the approach taken to calculate individual loss values is provided next.

**SimCLR** [7] is a contrastive Self-Supervised Learning method that uses a contrastive loss to learn representations from two augmentations of a set of images. The contrastive loss for a pair of positive samples $i, j$ is written as follows:

$$l_{i,j} = -log \frac{exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} exp(sim(z_i, z_k)/\tau)}. \tag{10}$$

The total loss for SimCLR is calculated by summing all the positive pairs in the batch [7]. Here, the sum of elements $l_{i,j}$, and $l_{j,i}$ corresponding to the augmentations of the same image is used as the loss term for the reducible anchor loss calculation.

**Barlow Twins** [31] uses a different loss function based on a cross-correlation matrix which computes a single combined output for all the samples in the batch without providing individual loss values. Given the mean-centered representations $Z^A$ and $Z^B$ from two augmented views of a batch of images, the cross-correlation matrix $C_{ij}$ is calculated as follows:

$$C_{ij} = \frac{\sum_b z^A_{b,i} z^B_{b,j}}{\sqrt{\sum_b \left(z^A_{b,i}\right)^2} \sqrt{\sum_b \left(z^B_{b,j}\right)^2}}, \tag{11}$$

Then the loss function for Barlow Twins $\mathcal{L}_{BT}$ is calculated as:

$$\mathcal{L}_{BT} = \sum_i \left(1 - C_{ij}\right)^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2 \tag{12}$$

It is not straightforward to extract individual loss terms from this loss function. In order to calculate reducible anchor loss, we divide the available batch of data into multiple smaller parts with $N_p$ images in each part $p$. Then we can calculate $\mathcal{L}_{BT}$ on these parts which are treated as a group and the selection of the whole group depends on the calculated reducible anchor loss of each part.

## 5 Experimental Results

In this section, we provide details about datasets, experiment settings, and implementation followed by results and various analyses on the mentioned datasets with our method. Additional details and experiments can be found in the supplementary material.

### 5.1 Datasets

**CIFAR-100.** This dataset consist of 60,000 small 32 × 32 images from 100 classes [14]. 50,000 of these images are used for training leaving the rest for testing. Although our method does not need to know the task boundaries, in order to have fair comparisons, we use the same 5-task class incremental setting as [10] for training. Therefore, the dataset is divided into 5 subsets of 20 classes and each chunk is fed in the same sequence to our model.

**ImageNet-100.** This dataset [29] is a 100 class subset of ImageNet [25] large-scale dataset. Training data consists of approximately 127k images (maximum of

1300 per class). 5000 images are also used for evaluation (50 per class). Similar to CIFAR-100, we feed the data in a 5-class class incremental setting to the models.

### 5.2 Settings

We follow the same setting as [10] to train and evaluate the models. All the models and baselines are trained with self-supervision on the training parts of the datasets. For the evaluation, the backbones are frozen and a linear model is trained on the outputs of the frozen backbone using the train set. Then the accuracy of the validation set is reported. In addition to our method, we conduct experiments using the CaSSLe [10] and an offline approach. To the best of our knowledge, the CaSSLe is the closest work to our method on CIFAR-100 and ImageNet-100 datasets. Our method has the disadvantage of being limited by the online setting but it also uses a larger batch of data in each step due to the use of memory. Therefore, a completely fair comparison is not possible and we mostly treat CaSSLe as a good base point to compare our approach.

The data of the first task in both datasets are used for offline pretraining in order to make comparisons with [10]. All the methods are initialized with the pre-trained weights of the offline model. In all the settings, training is executed for a varying number of steps to analyze the speed of convergence.

**Offline.** This setting is the regular training with access to the whole dataset. Since it does not have a method to prevent forgetting, we also include the data from the first task in this setting.

**CaSSLe.** The method proposed in [10] where the data from each task is processed multiple times separately without having access to the data from other tasks. A distillation mechanism is used to prevent forgetting.

**Ours.** Our proposed method in which the data is processed in an online approach. We include experiments with and without using RALS to analyze the effects of different parts of our work.

### 5.3 Implementation Details

We build upon the implementation provided by [10] which is itself based on [8]. The source code will be available at https://github.com/sinaazar/SpeedOSSL We adopt the same data preparation process as [10]. We use a batch size of 128 for streaming data. When training without RALS, the same 128 value is used to get a batch of data from the Memory. This value is increased to 256 when using RALS. While using RALS with Barlow Twins and SimCLR we choose $k = 256$ out of 384 samples in each step. Part size $N_p$ for Barlow Twins is set to 16 in all the experiments. We use ResNet18 [11] as the backbone model.

The maximum learning rate for the scheduler is set to 0.1 in ImageNet-100 experiments for both the Barlow Twins and SimCLR. On CIFAR-100, we use 0.05 and 1.0 for Barlow Twins and SimCLR, respectively. The initial learning rate is set to 0.1 of the maximum learning rate in all the settings. Maximum and minimum amounts for momentum are set to 0.95 and 0.85. All the experiments increase/decrease learning rate/momentum halfway through the training steps on each batch except the 50 steps experiments which the increase lasts for a quarter of the training steps. The default memory size for experiments on CIFAR-100 and ImageNet-100 is 10000 and 20000, respectively.

| Steps | Strategy | CIFAR-100 Acc | | ImageNet-100 Acc | |
|---|---|---|---|---|---|
| | | BT [31] | SCLR [7] | BT[31] | SCLR [7] |
| 10 | Offline | 48.5 | 50.68 | 44.74 | 58.94 |
| | CaSSLe [10] | 50.89 | 49.48 | 49.70 | 59.70 |
| | Ours Without RALS | 55.39 | 50.39 | 66.10 | 64.64 |
| | **Ours** | 55.75 | 50.63 | 66.94 | 64.86 |
| 20 | Offline | 56.23 | 51.71 | 67.94 | 67.84 |
| | CaSSLe [10] | 54.35 | 50.34 | 64.48 | 63.74 |
| | Ours Without RALS | 55.95 | 52.44 | 67.02 | 66.26 |
| | **Ours** | 56.64 | 52.77 | 68.18 | 66.78 |
| 50 | Offline | 59.64 | 54.29 | 69.24 | 71.26 |
| | CaSSLe [10] | 56.25 | 51.83 | 67.56 | 65.56 |
| | Ours Without RALS | 57.15 | 53.25 | 66.92 | 67.48 |
| | **Ours** | 58.48 | 54.76 | 68.40 | 67.98 |
| 100 | Offline | 62.93 | 56.64 | 74.66 | 73.28 |
| | CaSSLe [10] | 57.48 | 53.19 | 69.74 | 66.94 |
| Full | CaSSLe [10] | 58.26 | 55.91 | 69.90 | 67.96 |

Table 1: Comparison of results of SimCLR (SCLR) [7] and Barlow Twins (BT) [31] methods with various approaches and training steps on CIFAR-100 and ImageNet-100. The focus of this work is on a smaller number of steps but the longer trained versions of CaSSLe [10] are also reported for more insights.

### 5.4 CIFAR-100 Results

The results for CIFAR-100 [14] and ImageNet-100 dataset [29] datasets are presented in Table 1. Here, the linear evaluation accuracies for different strategies with a varying number of training steps are provided for SimCLR and Barlow Twins. Starting from SimCLR with 10 steps, our method outperforms CaSSLe trained for 10 and 20 steps with or without using RALS. This holds when the 20 steps version is compared with CaSSLe trained for 20 and 50 steps. Generally, increasing the number of steps shows similar improvements for our approach compared to CaSSLe. This compensates for the fact that an additional batch from memory is added to the streaming batch during the training of our method.

We observe similar results to SimCLR when training the model with the Barlow Twins method. Here, the difference in accuracy with CaSSLe becomes even larger in the 10-step setting. Again in all the step numbers, we see improvements over CaSSLe. It is also interesting to see that our method with RALS slightly outperforms the fully trained CaSSLe. The offline setting starts to dominate the results starting from 50 steps.

In all the experiments, including RALS leads to better performance than not including it albeit marginally in some scenarios. Generally, it can be observed that with more training steps we see higher improvements with RALS compared to only using the SLCLRS.

### 5.5 ImageNet-100 Results

Similar to CIFAR-100, we can see consistent improvements in all the training step numbers for both SimCLR and Barlow Twins. In all cases, using RALS leads to improvements compared to not using it. The improvements are more noticeable when using Barlow Twins. Based on these results and the reported accuracies from CIFAR-100, we can confidently validate the impact of RALS.

Similar to CIFAR-100 results, we see better performance from our method with RALS compared to CaSSLe in a higher step number setting. SimCLR results show the 50-step training version of our method performing on par with the fully trained version of the CaSSLe.

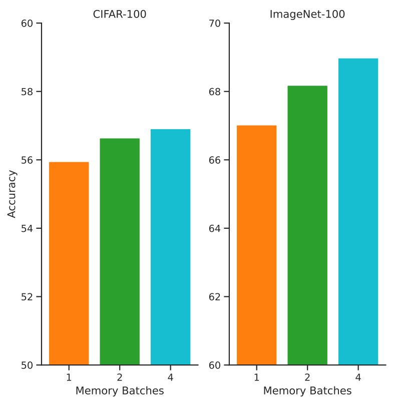
Overall, it can be seen that with steps as little as 10, noticeable results like 64.86% with SimCLR and 66.94% with Barlow Twins can be achieved in an online training scenario while using the ImageNet-100 dataset.

| Memory Size | SimCLR [7] Acc | Barlow Twins [31] Acc |
|---|---|---|
| 2000 | 50.43 | 55.68 |
| 5000 | 52.47 | 55.63 |
| 10000 | 52.77 | 56.64 |
| 20000 | 53.41 | 56.29 |

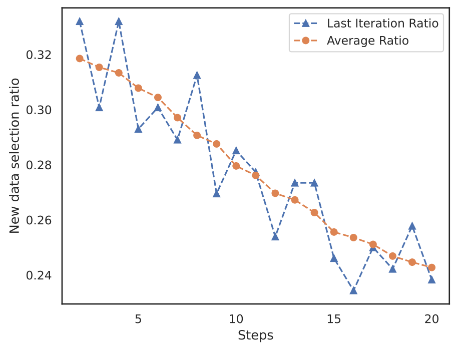
Table 2: Comparison of results of our approach with different backbone methods and memory sizes on CIFAR-100 while training the full model for 20 steps.

### 5.6 Analysis

**Memory.** The main goal of our method is to learn from a stream of data as fast as possible. Therefore, we are tackling a more challenging setup than offline Continual Self-Supervised Learning. Generally, it is not expected to be able to outperform offline CSSL. However, with a big enough memory, we observed improvements when only a small number of training steps are allowed. Table 2 shows the results on CIFAR-100 datasets for our full model trained for 20 steps with varying memory sizes. It can be observed that generally the bigger the memory the better the results.



(a)



(b)

Fig. 3: (a) Comparison of the accuracy of our method trained with different numbers of batches sampled from the memory in each iteration. The training was performed for 20 steps in each iteration with the Barlow Twins SSL algorithm on both CIFAR-100 and ImageNet-100 datasets. (b) Progression of selection ratio of the new batch to the combination of the new batch and samples drawn from the memory. The average ratio for the past 100 iterations in addition to the last iteration ratio for a 20-step training is shown here.

Considering the advantage of being online trainable, our approach is still interesting even with smaller memory size.

In Figure 3 (a), we can see the effect of increasing the number of batches sampled from the memory relative to the newly arrived batch of data. Based on the available training time between two iterations, the number of batches sampled from the memory can be increased to improve performance. The intuition behind it is that the RALS will have more samples to select from which in turn will lead to the selection of samples with higher reducible anchor loss.

**Selection Behaviour.** In order to get a better understanding of the distribution of the samples prioritized by the RALS during the local training steps, we keep track of the number of images selected from the new batch and the samples randomly selected from the memory. More specifically, we calculate the ratio of samples selected from the new batch to the total number of samples used in the current iteration. A visualization is provided in Figure 3 (b) where it shows the average ratio over a period of 100 iterations during training and the ratio of the last iteration. We can see that on average, during the initial steps of each local training session, the samples from the new batch are randomly selected. However, when the current model starts to deviate from the anchor model, the RALS starts to select samples more intelligently and it is easy to see that the more the new samples are used for training the less likely they are to be selected. This means the reducible anchor loss is indeed reducing for these samples and the model is learning quickly from them. It is also worth mentioning that the last iteration ratio is not necessarily monotonically decreasing like the average ratio which shows the flexibility of RALS based on the current distribution of samples.

## 6 Conclusion

In this work, we discussed the challenge of speeding up Online Self-Supervised Learning. We proposed using Single Local Cycle Learning Rate Scheduler to learn from the new samples of a stream of images as fast as possible. Additionally, the idea of Reducible Anchor Loss Selection was proposed to speed up training by selecting more useful samples. The results on both CIFAR-100 and ImageNet-100 with self-supervised learning methods of SimCLR and Barlow Twins showed consistent improvements on the speed of convergence.

## Acknowledgements

This work was partly supported by the Alexander von Humboldt Foundation.

## References

1. Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., Tuytelaars, T.: Memory aware synapses: Learning what (not) to forget. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 139–154 (2018)
2. Aljundi, R., Belilovsky, E., Tuytelaars, T., Charlin, L., Caccia, M., Lin, M., Page-Caccia, L.: Online continual learning with maximal interfered retrieval. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 11849–11860. Curran Associates, Inc. (2019), http://papers.nips.cc/paper/9357-online-continual-learning-with-maximal-interfered-retrieval.pdf
3. Aljundi, R., Kelchtermans, K., Tuytelaars, T.: Task-free continual learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11254–11263 (2019)
4. Aljundi, R., Lin, M., Goujaud, B., Bengio, Y.: Gradient based sample selection for online continual learning. Advances in neural information processing systems **32** (2019)
5. Buzzega, P., Boschini, M., Porrello, A., Abati, D., Calderara, S.: Dark experience for general continual learning: a strong, simple baseline. Advances in neural information processing systems **33**, 15920–15930 (2020)
6. Chaudhry, A., Ranzato, M., Rohrbach, M., Elhoseiny, M.: Efficient lifelong learning with a-gem. arXiv preprint arXiv:1812.00420 (2018)
7. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: International conference on machine learning. pp. 1597–1607. PMLR (2020)
8. da Costa, V.G.T., Fini, E., Nabi, M., Sebe, N., Ricci, E.: solo-learn: A library of self-supervised methods for visual representation learning. J. Mach. Learn. Res. **23**, 56–1 (2022)

9. Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A.A., Pritzel, A., Wierstra, D.: Pathnet: Evolution channels gradient descent in super neural networks. arXiv preprint arXiv:1701.08734 (2017)
10. Fini, E., da Costa, V.G.T., Alameda-Pineda, X., Ricci, E., Alahari, K., Mairal, J.: Self-supervised models are continual learners. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9621–9630 (2022)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
12. He, X., Jaeger, H.: Overcoming catastrophic interference using conceptor-aided backpropagation. In: International Conference on Learning Representations (2018)
13. Howard, J., Gugger, S.: Fastai: a layered api for deep learning. Information **11**(2), 108 (2020)
14. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
15. Lee, S.W., Kim, J.H., Jun, J., Ha, J.W., Zhang, B.T.: Overcoming catastrophic forgetting by incremental moment matching. Advances in neural information processing systems **30** (2017)
16. Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continual learning. Advances in neural information processing systems **30** (2017)
17. Madaan, D., Yoon, J., Li, Y., Liu, Y., Hwang, S.J.: Representational continuity for unsupervised continual learning. In: International Conference on Learning Representations (2021)
18. Mallya, A., Lazebnik, S.: Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. arXiv preprint arXiv:1801.06519 **6**(8) (2018)
19. Mindermann, S., Brauner, J.M., Razzak, M.T., Sharma, M., Kirsch, A., Xu, W., Höltgen, B., Gomez, A.N., Morisot, A., Farquhar, S., et al.: Prioritized training on points that are learnable, worth learning, and not yet learnt. In: International Conference on Machine Learning. pp. 15630–15649. PMLR (2022)
20. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
21. Purushwalkam, S., Morgado, P., Gupta, A.: The challenges of continuous self-supervised learning. In: Avidan, S., Brostow, G.J., Cissé, M., Farinella, G.M., Hassner, T. (eds.) Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXVI. Lecture Notes in Computer Science, vol. 13686, pp. 702–721. Springer (2022). https://doi.org/10.1007/978-3-031-19809-0_40, https://doi.org/10.1007/978-3-031-19809-0_40
22. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: icarl: Incremental classifier and representation learning. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 2001–2010 (2017)
23. Ritter, H., Botev, A., Barber, D.: Online structured laplace approximations for overcoming catastrophic forgetting. Advances in Neural Information Processing Systems **31** (2018)
24. Robins, A.: Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science **7**(2), 123–146 (1995)

25. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of computer vision **115**(3), 211–252 (2015)
26. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. arXiv preprint arXiv:1606.04671 (2016)
27. Serra, J., Suris, D., Miron, M., Karatzoglou, A.: Overcoming catastrophic forgetting with hard attention to the task. In: International Conference on Machine Learning. pp. 4548–4557. PMLR (2018)
28. Smith, L.N., Topin, N.: Super-convergence: Very fast training of neural networks using large learning rates. In: Artificial intelligence and machine learning for multi-domain operations applications. vol. 11006, pp. 369–386. SPIE (2019)
29. Tian, Y., Krishnan, D., Isola, P.: Contrastive multiview coding. In: European conference on computer vision. pp. 776–794. Springer (2020)
30. Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. arXiv preprint arXiv:1708.01547 (2017)
31. Zbontar, J., Jing, L., Misra, I., LeCun, Y., Deny, S.: Barlow twins: Self-supervised learning via redundancy reduction. In: International Conference on Machine Learning. pp. 12310–12320. PMLR (2021)
32. Zenke, F., Poole, B., Ganguli, S.: Continual learning through synaptic intelligence. In: International Conference on Machine Learning. pp. 3987–3995. PMLR (2017)