The Challenges of Continuous Self-Supervised Learning *(Supplementary Material)*

Senthil Purushwalkam[1⋆], Pedro Morgado[1,2⋆], and Abhinav Gupta[1]

[1] Carnegie Mellon University
[2] University of Wisconsin-Madison
www.senthilpurushwalkam.com/publication/continuousssl/

A Ethics and Societal Impact

Research on self-supervised learning has been making progress towards building systems that can continually learn from data in our world without human supervision. In this work, we propose a problem setup that evaluates the challenges faced when such methods are truly deployed in-the-wild. As these systems start being deployed without supervision, there are numerous possibilities for biases to emerge based on the statistics of data consumed. These biases could potentially have a negative impact on our society. Therefore, it is important to exercise caution when deploying such systems and relying on them for downstream applications. Before deploying such systems it is also important to thoroughly study and implement approaches to mitigate such emergent biases.

In our work, apart from working with existing datasets, we gather a collection of 100M images by downloading images from Flickr that have the Creative Commons license. While this license permits usage in our application, we do not plan to redistribute the images since they have not been thoroughly scanned for privacy concerns. The models trained in this work also have not been examined for potential societal biases or other spurious correlations that might have emerged from the data. While we plan to release the models trained here for research purposes, we would strongly advise against using them for any real-world applications.

B Implementation details

B.1 SimSiam

The experiments conducted in this paper make extensive use of SimSiam, a contrastive learning algorithm for self-supervised visual representation learning. We closely follow the official SimSiam implementation available at https://github.com/facebookresearch/simsiam.

⋆ Equal contribution.

In all experiments, we used ResNet-18 with synchronized batch-norm as the backbone. All models were trained on 4 GPUs using stochastic gradient descent (SGD) with a batch size of 256, learning rate of 0.05 with a cosine decay, momentum of 0.9, and weight decay of 0.0001. LIFO buffers are updated by always removing the oldest images. MinRed buffers are maintained by removing the most redundant images. Pseudo-code for the Buffered SSL with MinRed buffers is provided in Algorithm 1.

We follow the same evaluation protocol as in prior work, and use linear probes on the learned features to recognize classes of three datasets. The linear probes on ImageNet and iNaturalist were trained using the entirety of the datasets. On ImageNet, we trained the linear classifier using SGD+LARS for 10 epochs with a batch size of 1024, learning rate of 3.0 with cosine decay, momentum of 0.9, and no weight decay. On iNaturalist, the classifier was trained for 20 epochs with learning rate of 12.0. Contrarily to ImageNet and iNaturalist, evaluations conducted on the full ImageNet dataset ($14M$ images) only used a subset of the data (using only 50 images per class for training and 25 for evaluation). We trained the linear probe on this data using SGD+LARS for 30 epochs and a learning rate of 3.0.

B.2 Sampling and Splits for Lifelong Learning

In Section 6 of the main text, we construct a dataset with non-stationary semantic distributions to evaluate lifelong learning *i.e.* learning without forgetting. Here we describe the process of construction of this dataset.

First, we performed a depth-first search (DFS) on the Wordnet hierarchy. We split the sequence of DFS nodes (or classes) uniformly into four groups. Each such group contains classes that are close to each other in the Wordnet hierarchy and hence, semantically similar. In order to create the sequence of samples for lifelong learning, we could sequentially sample images from one split after the other. However, for future approaches, such hard boundaries in the sequence as we move from one split to the other could be easy to trivially identify and leverage to minimize forgetting. To make the setup more realistic, we create a smooth transition between one split to the other. The smooth transition is created by mixing the last 10% of each split with the first 10% of the next split. More concretely, we linearly decrease the likelihood of sampling from the first split and linearly increase the likelihood of sampling from the second split.

C Additional results

C.1 Generalization towards unseen categories

To assess the open set generalization ability of models trained with Minimum Redundancy (MinRed) buffers, we extended the continual learning experiment described in Section 6.2 and Figure 7 of the main paper, and further evaluate on future data partitions, *i.e.*, data partitions containing categories yet unseen

Algorithm 1: Buffered SSL with MinRed buffer. PyTorch pseudo-code.

```
def train(f, SimSiam, stream, num_updates):
  B = []  # Init empty buffer
  for ims in stream: # Load batch from stream
    Add2Buffer(B, ims)

    # Hyper-sampling: Update num_updates times
    for _ in range(num_updates):
      # Sample batch from buffer
      x = RandomSample(B)
      x1, x2 = aug(x), aug(x)
      z1, z2 = f(x1), f(x2)

      # Track features
      TrackRepresentations(B, x, (z1+z2)/2)

      # Compute loss and update models
      L = SimSiam(z1, z2)
      L.backward()        # Back-propagation
      update(f, SimSiam)  # SGD update

def Add2Buffer(B, ims):
  n_excess = len(B) + len(ims) - maxlen(B)
  if n_excess > 0:  # If full, remove n_excess.
    for _ in range(n_excess):
      # Pairwise dist
      d = pdist(B.feat, B.feat)

      # Distance to nearest neig
      d_nneig = d.min(dim=1)

      # Remove sample with smallest d_nneig
      i_redundant = d_nneig.argmin(dim=0)
      B.remove(i_redundant)

  # Add new images to buffer
  for x in ims:
    B.add(x)

def TrackRepresentations(B, x, z, alpha=0.5):
  # EMA update
  B.feat[x] = alpha*B.feat[x] + (1 - alpha)*z
```

in the training sequence. The results are shown in Fig. 1. Training models with MinRed buffers also lead to better generalizable towards unseen categories. This is likely explained by the fact that MinRed buffers maintain higher semantic

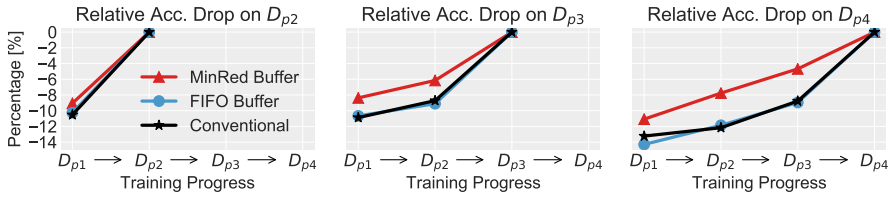

Fig. 1: Open set generalization. While training on the data stream used for assessing continual learning, we also evaluated the models on future data partitions, which contain images from images from categories not yet seen during training. By training models with MinRed buffers, we can learn representations the can better generalize to unseen categories.

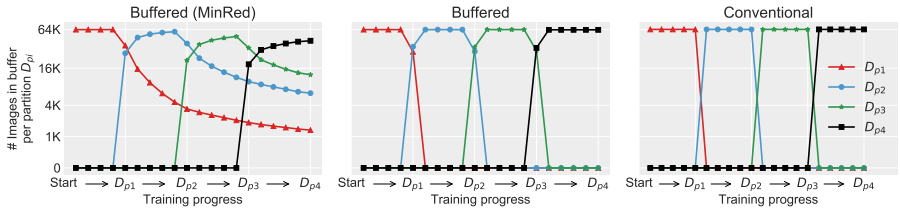

Fig. 2: Contents of 64K buffers as the data distribution shifts over the course of training. For Conventional SSL (which has no buffer), we count images in a sequence of training batches totaling the same 64K images. Buffered SSL with a Minimum Redundancy (MinRed) buffer retains a significant number of images from previous data distributions. This is in contrast to Buffered SSL with LIFO buffers or Conventional SSL which have no ability to retain images for long periods of time.

diversity in the training data, which encourages the model to learn more general representations, likely to generalize better to unseen categories.

C.2 Buffer contents during lifelong learning

To understand why MinRed buffers allow SimSiam to learn from non-stationary distributions with less forgetting (Section 6 of the paper), we analysed the contents of the buffer used to generate training samples. Figure 2 shows the number of images in the buffer from each of the $D_{p1}, \ldots, D_{p4}$ partitions, as training progresses from D_{p1} to D_{p4}. As can be seen, only MinRed buffers are capable of retaining images from prior distributions. Since these images can then be sampled for training, MinRed buffers enable continual training with less forgetting.

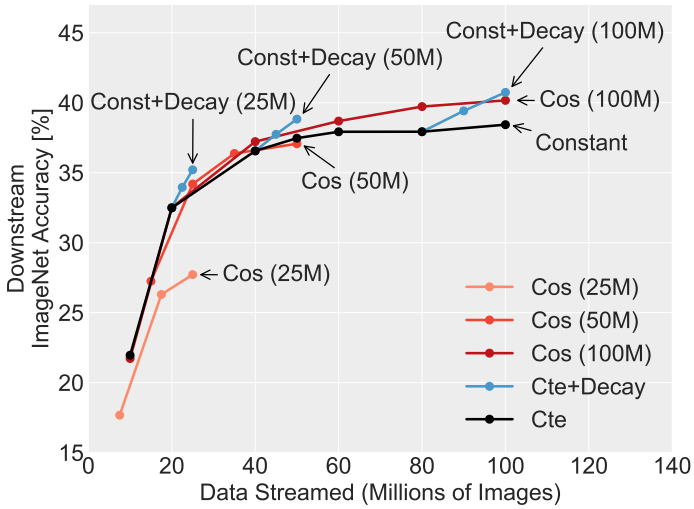

Fig. 3: Downstream performance on ImageNet throughout self-supervised training with various learning rate schedules. "Cos (xM)" stands for cosine decay ending at iteration x/batch size. "Const+Decay (xM)" represents a learning rate schedule with a constant start (for about 80% of the total training time), followed by a short cosine decay for the remainder of training.

C.3 Learning rate schedules for continual learning

The cosine learning rate schedule is not applicable to continuous SSL, as it requires a pre-determined end. We tested several learning rate schedules. Results are shown in Figure 3. With a simple constant learning rate, models can still learn from a continuous (non-stationary) data stream, while still being able to achieve similar performances in the static case, when combined with a short learning rate decay before evaluation.