A soft nearest-neighbor framework for continual semi-supervised learning
Zhiqi Kang, Enrico Fini, Moin Nabi, Elisa Ricci, Karteek Alahari

To cite this version:

HAL Id: hal-03893056
https://hal.science/hal-03893056v3
Submitted on 11 Sep 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
A soft nearest-neighbor framework for continual semi-supervised learning

Zhiqi Kang, Enrico Fini, Moin Nabi, Elisa Ricci, Karteek Alahari

Abstract

Despite significant advances, the performance of state-of-the-art continual learning approaches hinges on the unrealistic scenario of fully labeled data. In this paper, we tackle this challenge and propose an approach for continual semi-supervised learning—a setting where not all the data samples are labeled. A primary issue in this scenario is the model forgetting representations of unlabeled data and overfitting the labeled samples. We leverage the power of nearest-neighbor classifiers to nonlinearly partition the feature space and flexibly model the underlying data distribution thanks to its non-parametric nature. This enables the model to learn a strong representation for the current task, and distill relevant information from previous tasks. We perform a thorough experimental evaluation and show that our method outperforms all the existing approaches by large margins, setting a solid state of the art on the continual semi-supervised learning paradigm. For example, on CIFAR-100 we surpass several others even when using at least 30 times less supervision (0.8% vs. 25% of annotations). Finally, our method works well on both low and high resolution images and scales seamlessly to more complex datasets such as ImageNet-100. Our source code is publicly available at https://github.com/kangzhiq/NNCSL.

1. Introduction

Several efforts have been devoted to the continual learning (CL) [18] paradigm wherein training data samples arrive sequentially. However, most of the state-of-the-art CL methods [11, 12, 20] are based on a strong assumption: the data is fully labeled. This is an unrealistic requirement as labeling data is oftentimes expensive for the expertise required or the amount of annotations, hazardous due to privacy or safety concerns, or impractical in a real-time online scenario. A natural way of tackling this issue is by leveraging the semi-supervised learning framework, where not all the data samples are labeled.

In recent years, this stimulated the community to investigate a new line of research named continual semi-supervised learning [8,44,52]. It refers to the setting where each task in the sequence is semi-supervised. This learning scenario brings novel challenges, as the models catastrophically forget the representations of unlabeled data while also overfitting the labeled set. This is further exacerbated by another well-studied phenomenon in CL: overfitting the experience replay buffer [10]. These challenges result in vanilla CL methods underperforming, as they lack the ability to extract information from the unlabeled set, thus largely overfitting to the labeled set [53]. On the other hand, semi-supervised learning approaches [15,29,46,58] balance well the labeled and unlabeled sets but cannot handle the continual scenario, and suffer from forgetting even when paired with well-known CL methods (see Fig. 1 (right) and Tab. 1).

A few recent approaches [8,52] partially mitigate these issues on small-scale datasets. The method in [52] relies on generative models to replay previous classes, which leads to a sizeable computational and memory overhead, making it difficult to scale to larger datasets such as ImageNet. Boschini et al. [8] proposed to contrast among samples of different classes and tasks. However, [8] falls short with smaller memory buffers, larger datasets, more tasks, or higher resolution images (see Fig. 1 (left) and results in Sec. 6). There-
fore, we argue that there is a clear need for more powerful continual semi-supervised learning methods, with a more suitable use of the labeled set, and efficient and stable representation learning from the unlabeled set.

In this paper, we unleash the power of nearest-neighbors in the context of continual semi-supervised learning. In particular, we propose a new method, NNCSL (Nearest-Neighbor for Continual Semi-supervised Learning), that leverages the ability of the nearest-neighbor classifier to non-linearly partition the feature space in two ways: i) to learn powerful and stable representations of the current task using a self-supervised multi-view strategy, and ii) to distill previous knowledge and transfer the local structure of the feature space. The latter is achieved through our proposed NND (Nearest-Neighbor Distillation), a novel semi-supervised distillation loss that mitigates forgetting in continual semi-supervised learning better than other competitive distillation approaches. In contrast with knowledge distillation [11, 22, 30, 38] and feature distillation [19, 21, 24], which focus exclusively on class-level and sample-level distributions respectively, NND simultaneously distills relationships between classes and samples by leveraging the nearest-neighbor classifier. Overall, NNCSL outperforms all related methods by very large margins on both small and large scale datasets and both low and high resolution images. For instance, as shown in Fig. 1, NNCSL matches or surpasses all others with more than 30 times less supervision (0.8% vs. 25% of annotations) on CIFAR100.

The main contributions of this work are as follows:

- We propose NNCSL, a novel nearest-neighbor-based continual semi-supervised learning method that is, by design, impacted less by the overfitting phenomenon related to a small labeled buffer.

- We propose NND, a new distillation strategy that transfers both representation-level and class-level knowledge from the previously trained model using the outputs of a soft nearest-neighbor classifier, which effectively helps alleviate forgetting.

- We show that NNCSL outperforms the existing methods on several benchmarks by large margins, setting a new state of the art on continual semi-supervised learning. In contrast to previous approaches, our method works well on both low and high resolution images and scales seamlessly to more complex datasets.

2. Related work

Semi-supervised learning. Semi-supervised methods focus on learning models from large-scale datasets where only a few samples have associated annotations [17]. Early strategies for this learning paradigm applied to deep architectures leveraged pseudo-labels and performed self-training based on them [29]. This scheme was later improved with confidence thresholding [2] and adaptive confidence thresholding [56, 59]. More sophisticated methods for incorporating the confidence of the predictions and filtering out spurious samples were also developed, such as FixMatch [46], which employs a student-teacher architecture. Other approaches demonstrated the benefit of co-training [36] and distillation [55].

Another class of approaches was derived with the idea of imposing similar predictions from the network for two samples obtained with different input perturbations [3, 4, 27, 33, 35, 41, 47, 50, 60]. For example, [3] considered a consistency loss and soft pseudo labels generated by comparing the representations of the image views to those of a set of randomly-sampled labeled images. Recently, sample mixing techniques, such as MixUp, were also investigated in the context of semi-supervised learning for improving the model performance on low sample density regions [5, 6, 31]. However, none of the aforementioned works addressed the problem of learning in an incremental setting.

Continual learning. Several CL approaches have been proposed in the last few years to learn from data in an incremental fashion. According to a recent survey [18], existing CL methods can be roughly categorized into three groups. The first category comprises regularization-based methods, which address the problem of catastrophic forgetting by introducing appropriate regularization terms in the objective function [12, 19, 22, 24, 30] or identifying a set of parameters that are most relevant for certain tasks [13, 25, 54]. Replay-based methods correspond to the second group, and they store a few samples from previous tasks [9, 14, 32, 38] or generate them [34, 43] in order to rehearse knowledge during the training phases for subsequent tasks. Finally, the third category is parameter isolation methods [40, 42], which operate by allocating task-specific parameters.

While the vast majority of these methods operate in a supervised setting, recent works addressed the problem of overcoming catastrophic forgetting in the challenging case of limited [8, 28, 44, 52] or no supervision [1, 21, 37, 45]. However, most of them have default settings that are significantly different, e.g., the use of external datasets, and the accessibility of labeled/unlabeled data during continual learning stages, leaving only a few [8, 52] to be comparable in our desired realistic setting. Wang et al. [52] addressed the continual semi-supervised learning problem and proposed ORDIsCo, a method that continually learns a conditional generative adversarial network with a classifier from partially labeled data. However, the method yields prohibitive costs on higher resolution images, e.g., on ImageNet-100. Contrastive continual interpolation consistency (CCIC) [8] is another approach, which proposed to contrast samples among different classes and tasks. However, task-level contrastive learning showed insignificant improvement in the
ablation study. We speculate it may confuse the model as it gathers classes that are not necessarily related in the feature space. This confusion leads to worse performance with a small memory buffer, more samples, more tasks, or higher resolution images as shown by our experiments. Our work radically departs from these previous methods, as we design NNCSL, a novel approach for continual semi-supervised learning based on a soft nearest-neighbor classifier. Our empirical evaluation demonstrates that NNCSL surpasses prior works by a large margin.

3. Continual semi-supervised learning

We now formally define the problem of continual semi-supervised learning. Let the training data arrive sequentially, i.e., as a sequence of $T$ tasks. The dataset associated to task $t$ is denoted as $D_t$, with $t \in \{1, ..., T\}$. Learning is therefore performed task-wise, where only the current training data $D_t$ is available during task $t$. When switching from one task to the next one, previous data is systematically discarded. Since the available dataset is not fully labeled, we further divide it into two subsets such that $D_t = U_t \cup L_t$. Typically in a semi-supervised learning scenario, we have $|L_t| \ll |U_t|$, the ratio $|L_t|/|U_t|$ is kept constant for all the tasks. In addition, it is common practice in the CL literature [8, 38] to allow the retention of a memory buffer $M$ that stores and replays previously seen samples, as shown in Fig. 2.

Let $f_\theta$ be the model, parameterised by $\theta$, and consisting of three components: a backbone $g$, a projector $h$ and a classifier $p$. The backbone, here modeled as a convolutional neural network, is used to extract representations $z = g(x)$ from an input image $x$. The classifier takes this representation to predict a set of logits $p = p(z)$, while the projector (implemented as a multi-layer perceptron) maps the backbone features to a lower-dimensional space $h = h(z)$. In addition, we use superscript to refer to the state at a certain point in time, for instance for task $t$ as $f_\theta^t$, and for the previous task $t-1$ as $f_\theta^{t-1}$. Similarly, we use $x_u^t$ and $x_l^t$ to refer to samples drawn from $U_t$ and $L_t$ respectively. Apart from images, the labeled dataset also contains one-hot ground truth annotations $y$.

In the following sections, we introduce the proposed NNCSL method for continual semi-supervised learning. We first present PAWS [3], which inspired our NNCSL, (Sec. 4) and show why this method is not immediately applicable to the continual setting. Subsequently, we present CSL, our base continual semi-supervised learner (Sec. 5.1), which solves many of its issues. However, this base method lacks a mechanism to counteract forgetting. Hence, we introduce NND, our novel distillation approach based on the soft nearest-neighbor classifier in Sec. 5.2. All these elements are harmoniously integrated into our full method: NNCSL, whose overall objective is summarized in Sec. 5.3.

Figure 2: Illustration of the learning process in continual semi-supervised learning.

4. Nearest-neighbor meets continual learning: strengths and weaknesses

We now discuss the use of nearest-neighbor techniques [3] in the context of continual learning, and describe its strengths and weaknesses in scenarios with data distribution shifts. During training, the mini-batches that the model receives are composed of labeled and unlabeled data, with $K$ and $N$ as batch sizes for these two sets respectively. Unlabeled images in the batch are augmented twice using common data augmentation techniques to obtain two correlated views of the same sample $(x, x)$. The model processes the batch, producing the projected representations $h_u$ and $(h_u^l, h_u^l)$ for labeled and unlabeled samples respectively.

The main idea from [3] is to assign pseudo-labels for unlabeled samples in a non-parametric manner by considering their relationship with labeled samples, i.e., nearest-neighbor label assignments. Samples are compared in the feature space using the cosine similarity of projected features, and then the pseudo label is obtained by aggregating labels according to the similarities. More formally, let the superscript $k$ represent the index of the $k^{th}$ sample in the labeled mini-batch, and $\delta$ be the cosine similarity. One can apply a soft nearest-neighbor classifier to classify the augmented unlabeled sample $x_u$ as follows:

$$\hat{y} = \text{SNN}(\hat{h}_u, S, \epsilon) = \sum_{k=1}^{K} \frac{e^{\epsilon \delta(h_u, h_u^k)/\epsilon}}{\sum_{i=1}^{K} e^{\epsilon \delta(h_u, h_u^i)/\epsilon}} y^k,$$

where $S = [h_1^l, ..., h_K^l]$ are the features of the support samples and $\epsilon$ is a sharpening parameter that controls the entropy of the pseudo-label. Similarly, we can classify the other view of the same sample $v = \text{SNN}(\hat{h}_u, S, \tau)$, with the only difference that we use a more gentle sharpening parameter $\tau > \epsilon$, referred to as the temperature. Now, we can use $\hat{v}$ as a target pseudo-label and train the network through the cross-entropy loss:

$$L_{\text{SNN}} = H(v, \hat{v}).$$
to output consistent representations for similar inputs, while also accounting for the distribution of the classes in the feature space. However, one issue with this formulation is that the network could output unbalanced or even degenerate predictions where some classes are predicted more frequently than others. To avoid this, PAWS imposes the distribution-wise likelihood of all the classes to be uniform using a regularization term called Mean Entropy Maximization (MEM) loss defined as:

$$L_{MEM} = H\left(\frac{1}{N} \sum_n \hat{y}_n\right).$$

(3)

Given these two losses, the total loss for PAWS is a weighted average of the two.

The advantage of this soft nearest-neighbor formulation is that it utilizes labeled samples as support vectors, not as training samples, which reduces overfitting. This property is interesting from the point of view of continual learning, since we would like to extract as much training signal as possible from the memory buffer without overfitting to it. However, PAWS is not designed to work under data distribution shifts. The key issue of PAWS in the CL setting is the assumption that the labeled and unlabeled sets exhibit the same distribution. This is untenable in CL, as the memory buffer contains classes not in the current task’s unlabeled set. MEM loss aggravates this problem, as it tries to scatter the pseudo label over all the classes, even for the ones whose unlabeled samples are unavailable. A simple solution would be to use the labeled data of the current task and discard the buffer, but this is sub-optimal as the buffer is critical for CL.

Another important drawback of PAWS in the context of CL is the ambiguity of two-stage methods. PAWS performs best when the learned backbone is fine-tuned using the labeled set. This two-stage pipeline (pre-training and then fine-tuning) is prone to a loss in generalization performance as the labeled set is very small. Offline methods such as PAWS can afford this loss to gain specialized knowledge on the targeted tasks. However, CL requires the reuse of the model for the next task. On one hand, using the fine-tuned model leads to a noticeable loss in performance on the subsequent tasks. On the other hand, using the pre-trained checkpoint (before fine-tuning) preserves more general properties but brings additional memory overhead, as two models need to be stored, which is undesirable in CL. In the following section, we describe our solution to overcome these limitations. A detailed illustration of the ambiguity of two-stage methods can be found in the supplementary material.

\[1\] With a slight abuse of notation we refer to \(H(\cdot)\) with one argument as the entropy function, while when two arguments are passed we consider it as the cross-entropy function \(H(\cdot, \cdot)\).

Figure 3: Overview of the base learner component of our method, which does not have a distillation loss. We refer to this as CSL.

5. NNCSL: Our nearest-neighbor approach for continual semi-supervised learning

5.1. Our continual semi-supervised learner

We now describe our proposed approach that leverages the strengths of the nearest-neighbor approach described in Sec. 4, while overcoming its weaknesses. The easiest way to make the labeled and unlabeled distributions match is to disregard the memory buffer. This is clearly undesirable for CL. However, one could multi-task PAWS with another objective that also takes into account the information of the memory buffer. In particular, we suggest processing labeled samples of all the classes seen so far, but filtering out samples from the previous tasks so they do not interfere in the computation of Eq. 1. However, we can use the output of the linear classifier \(p\) and optimize a standard cross-entropy loss:

$$L_{LIN} = \sum_j H(p^k, y^j),$$

(4)

on all the \(J\) labeled samples in the current batch (which also contains \(K\) labeled samples of the current task). The complete loss for our base continual semi-supervised learner (named CSL) is as follows:

$$L_{CSL} = L_{SNN} + \lambda_{MEM} \cdot L_{MEM} + \lambda_{LIN} \cdot L_{LIN}.$$ 

(5)

This loss has several favorable effects; it: i) stimulates the network to focus on the old classes while learning representations of the new ones through PAWS, ii) creates an ensemble effect between the two classifiers, iii) completely removes the need for fine-tuning, as the linear classifier is trained online, and iv) enables us to control the trade-off between fitting labeled or unlabeled data through the parameter \(\lambda_{LIN}\). Interestingly, we found that very small values of \(\lambda_{LIN}\) work well in practice, while larger values in-
To mitigate forgetting, we use the probabilities predicted by the teacher for an unlabeled sample and input by the teacher for an unlabeled sample. This equates to computing the following two vectors:

\[ \mathbf{w} = \text{SNN}(\mathbf{h}_u, \mathbf{R}, \tau) \quad \text{and} \quad \mathbf{w}^{t-1} = \text{SNN}(\mathbf{h}_u^{t-1}, \mathbf{R}_t^{t-1}, \tau), \]

where \( \mathbf{h}_u^{t-1} \) is a feature vector output by the teacher for an unlabeled sample \( x_u \), while \( \mathbf{R} \) and \( \mathbf{R}_t^{t-1} \) represent the support set of previous classes embedded in the old and new feature spaces respectively. To mitigate forgetting, we use the probabilities predicted by the teacher as a distillation target:

\[ \mathcal{L}_{\text{NND}} = H(\mathbf{w}, \mathbf{w}^{t-1}). \]  

Note that the output of the teacher is not sharpened as it is done in Eq. 1. We apply the same temperature for both new and old features. We emphasize that here we use an inverted filter as that of Sec. 5.1, to distill knowledge about the previous classes only. See Fig. 4 for a visual intuition and supplementary material for the visualization of its impact on deep features. It is worth mentioning that the memory overhead introduced by storing \( f_0^{t-1} \) can be easily reduced by storing instead the features, which are lightweight compared to images. For instance, storing a 128-dimension feature takes roughly \( 1/1000^{th} \) memory w.r.t. an RGB image of size 224 \( \times \) 224.

Our NND loss is different from the standard distillation loss. Knowledge distillation focuses on class-level distributions calculated by comparing samples of the new tasks with the prototype of each class. This causes a loss of information about representations of individual samples in the latent space. On the contrary, NND distills the aggregated relationships of each unlabeled sample with respect to all the labeled ones in the mini-batch. This encourages the model to maintain more stable representations, by anchoring unlabeled samples to labeled samples. In addition, NND can capture and transfer non-linear sample-class relationships by definition, unlike knowledge distillation which is limited to linear boundaries. Also, the SNN classifier is computed on a different support set sampled from the buffer at every iteration, which introduces randomness that further regularizes the model. Furthermore, when distilling at very low temperatures (e.g., 0.1), the \textit{softmax} function from Eq. 1 behaves like an \textit{argmax} (i.e., selecting the closest samples in the feature space with high cosine similarity), leading to the transfer of information mainly on the local neighborhood from the teacher to the student. Finally, with respect to feature distillation which lacks alignment with class labels, NND carries more information about class distributions, which results in improved performance.

5.3. Overall loss

The overall NNCLS model, composed of two novel components, i.e., CSL and NND, is trained with the loss:

\[ \mathcal{L}_{\text{NNCLS}} = \mathcal{L}_{\text{CSL}} + \lambda_{\text{NND}} \cdot \mathcal{L}_{\text{NND}}. \]  

6. Evaluation and analysis

6.1. Experimental settings

\textbf{Datasets.} We evaluate our method on three datasets. CIFAR-10 [26] is a dataset of 10 classes with 50k training and 10k testing images. Each image is of size \( 32 \times 32 \). CIFAR-100 [26] is similar to CIFAR-10, except it has 100
the LARS optimizer [57] with a momentum value of 0.9.

As in [8], we use ResNet18 as our backbone for all the datasets. We adopt the implementation of [3], unless explicitly stated. Specifically, we use the LARS optimizer [57] with a momentum value of 0.9.

We set the weight decay as $10^{-5}$, the batch-size as 256. The learning rate is set to 0.4 for CIFAR-10 and 1.2 for CIFAR-100 and ImageNet-100, respectively. We apply 10 epochs warm-up and reduce it with a cosine scheduler. For the two correlated views of the same sample, we generate two large crops and two small crops of each sample. The large crops serve as targets for each other, whereas they are both targets for the small crops. We apply data-augmentation as in [16]. Label smoothing is applied with a smoothing factor of 0.1. The additional linear evaluation head is a simple linear layer, which we use to predict labels at test time. We choose $\lambda_{NND} = 0.2$ and $\lambda_{LIN} = 0.005$. As for the memory buffer, we utilize a simple random sampling strategy. We run our experiments 3 times with different random seeds. The standard deviation is also reported, if applicable. Further implementation details and analysis of data augmentation can be found in the supplementary material.

**Continual semi-supervised setting.** For both CIFAR-10 and CIFAR-100, we train the models with three different levels of supervision, i.e., $\lambda \in \{0.8\%, 5\%, 25\%\}$. For instance, this corresponds to 4, 25, 125 labeled samples per class in CIFAR-100. As for ImageNet-100, we opt for 1% labeled data. To build the continual datasets, we use the standard setting in the literature [8, 52], and divide the datasets into equally disjoint tasks: 5/10/20 tasks for CIFAR-10/CIFAR-100/ImageNet-100, i.e., 2/10/5 classes per task, respectively. We follow the standard class-incremental learning setting in all our experiments: during the CL stages, we assume that all the data of previous tasks are discarded. A memory buffer can be eventually built, but only for labeled data. Following [8], we set the buffer size for labeled data as 500 or 5120, to ensure a fair comparison.

**Baselines.** As baselines, we first consider traditional fully-supervised CL methods. A straightforward way to convert them into a continual semi-supervised setting is to use only the labeled data during training and to discard the unlabeled data. We consider two categories of methods: replay-less methods, namely, continually fine-tuning the model on each new task, online elastic weight consolidation (oEWC) [25], and replay-based methods, i.e., experience replay (ER) [39], iCaRL [38], FOSTER [51] and X-DER [7]. We denote PseudoER as an additional two-stage baseline method, which is a combination of semi-supervised learning (PAWS) and CL (ER) methods. More

### Table 1: Average accuracy with standard deviation of different methods tested with 5-task CIFAR-10 and 10-task CIFAR-100 settings. The number between brackets indicates the size of the memory buffer for the labeled data.

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.8%</td>
<td>5%</td>
</tr>
<tr>
<td>Fine-tuning</td>
<td>13.6±2.9</td>
<td>18.2±0.4</td>
</tr>
<tr>
<td>oEWC [25]</td>
<td>13.7±1.2</td>
<td>17.6±1.2</td>
</tr>
<tr>
<td>ER (500) [39]</td>
<td>36.3±1.1</td>
<td>51.9±4.5</td>
</tr>
<tr>
<td>iCaRL (500)</td>
<td>24.7±2.3</td>
<td>35.8±3.2</td>
</tr>
<tr>
<td>FOSTER (500)</td>
<td>43.3±0.7</td>
<td>51.9±13</td>
</tr>
<tr>
<td>X-DER (500)</td>
<td>33.4±1.2</td>
<td>48.2±1.7</td>
</tr>
<tr>
<td>PseudoER (500)</td>
<td>50.5±0.1</td>
<td>56.5±0.6</td>
</tr>
<tr>
<td>CCIC [8] (500)</td>
<td>54.0±0.2</td>
<td>63.3±1.9</td>
</tr>
<tr>
<td>PAWS [3] (500)</td>
<td>51.8±1.6</td>
<td>64.6±0.6</td>
</tr>
<tr>
<td>CSL (500)</td>
<td>64.5±0.7</td>
<td>69.6±0.5</td>
</tr>
<tr>
<td>NNCSL (500)</td>
<td>73.2±0.1</td>
<td>77.2±0.2</td>
</tr>
<tr>
<td>PseudoER (5120)</td>
<td>55.4±0.5</td>
<td>70.0±0.3</td>
</tr>
<tr>
<td>CCIC [8] (5120)</td>
<td>55.2±1.4</td>
<td>74.3±1.7</td>
</tr>
<tr>
<td>ORDisCo [52] (12500)</td>
<td>41.7±1.2</td>
<td>59.9±1.4</td>
</tr>
<tr>
<td>CSL (5120)</td>
<td>64.3±0.7</td>
<td>73.1±0.3</td>
</tr>
<tr>
<td>NNCSL (5120)</td>
<td>73.7±0.4</td>
<td>79.3±0.3</td>
</tr>
</tbody>
</table>

For labeled data as 500 or 5120, to ensure a fair comparison. The standard deviation is also reported, if applicable.
precisely, we continually train a PAWS and use it to self-label the unlabeled data. An ER method is applied afterward on the labeled and pseudo-labeled data. We also consider continual semi-supervised learning baseline methods, such as CCIC [8] and ORDisCo [52]. While CCIC has an explicit definition of the memory buffer for labeled data, which is either 500 or 5120, ORDisCo directly stores all the labeled data that the model receives. We thus denote its memory buffer size with the largest value, which is \( M = 12500 \), equivalent to 25% of CIFAR-10. As for the upper bound, which is commonly shown in CL [18], it is obtained by jointly training the model with all the available data. The lower bound refers to continual fine-tuning. CSL is also included as an ablation of NNCSL without NND loss.

### 6.2. Results

**CIFAR-10 and CIFAR-100.** We first report the performance of different methods on CIFAR-10 and CIFAR-100 in Tab. 1. The upper bound on CIFAR-10 is 92.1±0.1%, and that on CIFAR-100 is 67.7±0.9%. NNCSL outperforms all the competitors in all settings but one, with a significant margin. For instance, when using a buffer of 5120 and 0.8% of labeled data, NNCSL performs better than or substantially matches almost all the others, even when they use 25% labeled data, i.e., about 30 times more supervision. It is also interesting to note that NNCSL has a very low variance (\( \leq 0.7 \)) across all the settings, indicating a better convergence and representation learning during training.

From the results in Tab. 1, the memory buffer is shown to be effective to alleviate forgetting, as replay-based methods significantly outperform regularization-based ones. PseudoER performs better than ER when labeled data is limited (0.8%), but underperforms when the ratio is higher (5%, 25%). We conjecture this as a result of noisy pseudo-labeled data replacing the ground truth in the memory buffer by the sampling strategy of ER, which causes stronger forgetting. In addition, PseudoER is upper-bounded by PAWS, since ER is dependent on the accuracy of pseudo-labeling.

Methods such as CCIC and ORDisCo\(^1\) benefit from their design specific to the continual semi-supervised learning scenario. Even though ORDisCo has a larger memory buffer, its performance is inferior to that of CCIC. We believe that this is due to the difficulty in jointly training a continual classifier with a GAN model. CCIC performs reasonably well on CIFAR-10, especially with a large memory buffer. When the buffer size is 5120, CCIC performs better than NNCSL in the 25% setting. We suspect that our method underfits in this setting, due to the very small weight of the linear evaluation loss. In contrast, CCIC relies more on labeled data, with equal or even higher importance for the supervised loss. To validate our hypothesis, we increased the weight \( \lambda_{\text{LIN}} \) for our linear classifier loss. We obtained an accuracy of 84.5±0.4%, which is comparable with CCIC in the same setting (84.7±0.9%). In the case of a dataset with more classes such as CIFAR-100, the superiority of NNCSL is clearly evident. Further comparison with the replay strategy of ORDisCo can be found in the supplementary material.

**ImageNet-100.** We also evaluate the more challenging benchmark of ImageNet-100 in a 20-task continual semi-supervised setting and buffer size 5120. As shown in Tab. 2, NNCSL is the best-performing method. CCIC does not show significant improvement with more labeled data (5% & 25%) and also becomes inferior to vanilla CL methods, e.g., ER. We suspect this is a limitation of its representation learning for extracting information from images with higher resolution and larger variance, as we observe its training and validation accuracy in this setting is significantly lower. In addition, Fig. 5 illustrates the evolution of average accuracy. We note that the average accuracy of our NNCSL stabilizes at around 30% after the 10\(^{th}\) task and has a clear rebound between tasks 11 and 16, showing that NNCSL can effectively retain knowledge acquired during the continual learning steps. In contrast, previously competitive methods clearly suffer from forgetting in this challenging setting, especially in the first several tasks. Despite the same rebounding effect in some methods, e.g., FOSTER, they fail to remain close to NNCSL.

### 6.3. Additional analyses

**Ablation.** We ablate the components of our framework in Tab. 3 on both CIFAR-10 (C10 in the table), CIFAR-100 (C100) and ImageNet100 (IN100). The largest improvement comes from filtering (10.8% and 6.2% improvements on CIFAR-10 and CIFAR-100 respectively) and distillation (9.4% and 4% improvements on CIFAR-10 and CIFAR-100 respectively). This confirms the contributions of our proposed components. Although the linear evaluation loss

---

\(^1\)Note that the results of ORDisCo are directly provided by the authors of [52] as there is no open-source implementation of their approach.
Figure 5: Comparison of NNCSL with existing methods on 20-task Imagenet-100 and 1% of labeled data. The average accuracy after each learning step is shown.

Table 3: Ablation study of the effectiveness of the proposed components on 5-task CIFAR-10 and 10-task CIFAR-100, with \( M = 500 \) and 0.8% of labeled data, and 20-task ImageNet-100 with \( M = 5120 \) and 1% of labeled data. We use average accuracy as metrics.

Table 4: Final accuracy on each task after training on 5-task CIFAR-100. We use CSL as baseline to which we add different distillations. NNCSL is equivalent to CSL + NND.

Figure 6: Average training accuracy of unlabeled data on 5-task CIFAR-10. Comparison between vanilla PAWS and our proposed CSL in the continual semi-supervised setting.

Table 5: Training and validation accuracies on the current task (i.e., evaluate on task \( t \) while training on task \( t \)) with different values of \( \lambda_{\text{LIN}} \). Larger weights increase overfitting on the labeled set and reduce generalization. These experiments are conducted on CIFAR-10 with 5 tasks.

**Evidence of effectiveness.** We visualize in Fig. 6 the learning curve of average accuracy for unlabeled training data. As we illustrate in Sec. 5.1, the vanilla MEM loss is detrimental to the learning as it forces the model to assign incorrect pseudo-labels to the unlabeled data. Our proposed CSL effectively resolves this issue and allows the model to learn a better representation with the unlabeled data. Note that we only use the labels of unlabeled data to monitor the training, and no label information is leaked at train time.

**Linear evaluation head.** We report the train and validation accuracy in Tab. 5. Note that for the accuracy at train time, we use the average of accuracy on the current task, i.e., evaluation on task \( t \) while training on task \( t \). In general, having a high training accuracy of unlabeled data means the model learns a good representation, which leads to a good validation accuracy. One can observe overfitting when \( \lambda_{\text{LIN}} \) grows. This justifies our choice of i) having a small weight for the linear evaluation head, and ii) choosing PAWS as the basis, which does not directly use labeled data as training samples to avoid overfitting. Moreover, we observe underfitting when \( \lambda_{\text{LIN}} \) is smaller than 0.005. It confirms the need for this linear classifier in the continual semi-supervised scenario. We thus set \( \lambda_{\text{LIN}} \) as 0.005 in our experiments.

**Ablation of memory buffer.** To verify the effectiveness of the replay buffer, we evaluate our NNCSL without this and observe a drastic decrease in performance (19.7% vs 73.2%) on 5-task CIFAR-10 with 0.8% of labeled data. However, our method recovers 95% of the performance.

---

**Impact of distillation.** Complementary to Fig. 1 (right), which qualitatively shows the superior performance of our proposed NND, we report in Tab. 4 the final accuracy of each task after training on all tasks. An effective distillation method should maintain the performance of old tasks (e.g., task 1 for NND vs. feature distillation) and efficiently learn new tasks (e.g., task 5 for NND vs. knowledge distillation).
while using only 10% of memory (50 vs. 500). We conjecture such data efficiency results from the linear classifier that provides strong gradients in our method. More details of this study can be found in the supplementary material.

7. Conclusion

In this work, we studied continual semi-supervised learning and proposed NNCSL, a novel approach based on soft nearest-neighbors and distillation. Our extensive experiments show the superior performance of NNCSL with respect to existing methods, setting a new state of the art. Previous work [3] showed that using a more powerful network such as wider or deeper ResNet can further improve performance. While this is not addressed in this work, we consider it an interesting direction for future work. In this paper, we considered a fixed ratio between labeled and unlabeled samples across all tasks. A varying ratio would be an even more challenging setting for future investigation.

Acknowledgements This work was funded in part by the ANR grant AVENUE (ANR-18-CE23-0011). It was also granted access to the HPC resources of IDRIS under the allocation 2021-[AD011013084] made by GENCI.

Appendix

A. Implementation details

Although our model shares most of the hyper-parameters across different datasets, there are few differences, in values chosen empirically, to adapt to different scenarios. NNCSL, as well as PAWS [3] and CSL for ablation study, are trained with 250 epochs per task for CIFAR-10 and CIFAR-100, and 100 epochs for ImageNet-100. For CIFAR-10, the learning rate is initialized as 0.08, warmed up to 0.4, and reduced to 0.032 with the cosine scheduler. For CIFAR-100, a similar variation of learning rate is set from 0.08 to 1.2 to 0.032, and for ImageNet-100, it is 0.3 to 1.2 to 0.064. The color distortion ratio is set to 1 for ImageNet-100 and 0.5 for CIFAR-10 and CIFAR-100. The size of the mini-batch for labeled data is set to 5 for CIFAR-10 and 3 for CIFAR-100 and ImageNet-100. The size of the mini-batch for unlabeled data is set to 256 for CIFAR-10 and CIFAR-100 and 64 for ImageNet-100. These hyper-parameters are mostly based on the suggested default values of PAWS, and we empirically update them after testing with a moderate set of values variant around the default ones, based on the validation performance. However, We do not perform hyper-parameter tuning on ImageNet-100: we first adopt the hyper-parameters for ImageNet from PAWS and update them with the same changes we apply on CIFAR-100.

For the continual learning setting, we initialize a unified linear evaluation head where the number of outputs is the total number of classes in the dataset. When a class is not yet seen by the model, the corresponding output is masked. To retain a copy of the previously trained model, we use the `deepcopy` method from the `copy` package.

The copied model is in evaluation mode when training the current model on the new classes.

We have included our source code as part of the supplementary material. All the implementation details can be found in the options files, for instance, random seeds, and labeled samples on each dataset. We plan to open-source our code upon acceptance of this submission.

B. Comparison of data augmentation

We note that the data augmentation of our proposed framework is not the same as the one used in CCIC [8]. CCIC utilizes random cropping and horizontal flipping (which we refer to as weak DA), whereas our proposed CSL and NNCSL include color distortion as an additional operation for data augmentation (referred to as strong DA). To verify the impact of this additional augmentation strategy, we include color distortion in the data augmentation process of CCIC and re-train it from scratch on CIFAR-10 (C10 in the table) with 5 tasks, 5% labeled data and buffer size 5120, and also on CIFAR-100 (C100 in the table) with 10 tasks, 0.8% labeled data and buffer size 5120. The results are reported in Tab. 6. CCIC does not benefit from color distortion on both datasets. We believe this is because CCIC does not have the multiple-view consistency to be robust with respect to strong data augmentation. Consequently, we chose to report results using CCIC’s original (and more effective) data augmentation.

<table>
<thead>
<tr>
<th>Method</th>
<th>Dataset</th>
<th>Data Augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Weak</td>
</tr>
<tr>
<td>CCIC</td>
<td>C10</td>
<td>72.8</td>
</tr>
<tr>
<td>CCIC</td>
<td>C100</td>
<td>12.0</td>
</tr>
</tbody>
</table>

Table 6: Comparison of different data augmentation strategies for CCIC on CIFAR-10 (denoted as C10 in the table) and CIFAR-100 (denoted as C100 in the table).

<table>
<thead>
<tr>
<th>Method</th>
<th>Replay strategy</th>
<th>Average Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNCSL</td>
<td>Labeled</td>
<td>76.7</td>
</tr>
<tr>
<td>NNCSL</td>
<td>Labeled &amp; Unlabeled</td>
<td>82.1</td>
</tr>
<tr>
<td>ORDisCo</td>
<td>Labeled &amp; GR</td>
<td>65.9</td>
</tr>
</tbody>
</table>

Table 7: Comparison of different strategies for the replay buffer with 5-task CIFAR-10, using 3% of labeled data to match the setting of [52].

†https://docs.python.org/3/library/copy.html
predict

T2

T3

Table 8: CIFAR-10 average accuracy wrt memory buffer sizes with 5 tasks, 0.8% of labeled data.

C. Replay strategies

ORDisCo [52] utilizes a generative replay (GR) strategy to replay unlabeled data. Given that the generative model brings a memory overhead that is not negligible, it is reasonable to equip our method with a memory buffer for unlabeled data for a fair comparison. Specifically, we use 5000 samples, which is equivalent to the size of the generative model of ORDisCo. Tab. 7 shows that having access to the previously seen unlabeled data can indeed improve the performance of our method, and our NNCSL performs better with a simple memory buffer than ORDisCo with a sophisticated generative-replay strategy. This experiment confirms the ability of our method to exploit unlabeled data.

D. Ablation study of the memory buffer

We present the ablation study of the memory buffer in a table, as shown in Tab. 8, for better readability. The use of a memory buffer is a common practice in CL and most replay-based methods would fail if the buffer is removed. For instance, we observe a drastic decrease in performance (19.7% vs. 73.2%) when no replay buffer is allowed. In addition, increasing the memory buffer leads to improvements in performance with a certain upper bound. The performance plateaus when the memory buffer is larger than 500 (500 vs. 5000) because the buffer size is larger than the total number of labeled data in this case (i.e., 400 labeled samples).

E. Ambiguity of two-stage methods in CL

We show in Fig. 7 how two-stage methods (pre-training and then fine-tuning) can be adapted to a CL setting. For instance, after training on Task 1, one has to choose the model to be utilized in the subsequent tasks. The left path reuses the pre-trained model. It ensures the generalizability for learning each new task but results in additional memory overhead, as the fine-tuned model has to be saved for testing. In contrast, the right path reuses the fine-tuned model, which leads to a unified model for all tasks. However, the overfitting caused by the small labeled set is detrimental to the generalizability of the model. We explicitly show such loss in Fig. 7 by associating the size of the fine-tuned model with its generalizability. On the right path, the fine-tuned model is shrinking due to the overfitting issue.

F. Training analysis on ImageNet-100

It is interesting to see that PAWS diverges in this setting, as is shown in the Tab. 3 of the main paper. Our analysis reveals that the vanilla MEM loss strongly impacts representation learning on unlabeled data and makes the learning procedure highly unstable, as shown in Fig. 8. Although we do not observe the same collapse of PAWS on CIFAR-10 or CIFAR-100, recall that in Fig. 6 of the main paper, the training accuracy of unlabeled data for PAWS is strongly constrained on CIFAR-10. This means the representation learning of PAWS is already vulnerable. As images of ImageNet-100 have a much larger resolution than that of CIFAR-10, learning a robust feature from the input of ImageNet-100 is significantly more difficult. In such a complex case, the model easily diverges but can hardly recover, we suspect that it is because the gradient is very noisy (due to the negative impact of MEM loss) and small (due to the partial supervision and indirect use of labeled data). To verify this assumption, we observe that adding the linear head slightly alleviates the divergence. However, it cannot prevent the collapse from happening. This means that the MEM loss is the main cause of the collapse and is indeed detrimental to representation learning.

Nevertheless, we believe it may be possible to resolve this collapse without changing the framework, i.e., PAWS. For example, one can conduct careful, extensive hyperparameter tuning to find an optimal set of parameters that can stabilize the learning. However, it is not realistic given the scale of the dataset. Hence, we did not conduct such experiments.

G. Impact of \( \lambda_{NND} \)

In Tab. 9, we report the performance of our NNCSL with respect to different values of \( \lambda_{NND} \) on CIFAR-100, with 5 tasks, 1% of labeled data and buffer size 5120. \( \lambda_{NND} \) controls the importance of the distillation branch with respect to the PAWS loss. The higher the value, the stronger constraint the model receives to retain the previous knowledge. \( \lambda_{NND} = 0 \) means no distillation, which reduces the model back to CSL. We can clearly see that distillation helps the
model perform better (e.g., $\lambda_{\text{NND}} = 0$ vs. $\lambda_{\text{NND}} = 0.2$) and too much regularization from distillation can constrain the model from learning new knowledge (e.g., $\lambda_{\text{NND}} = 0.2$ vs. $\lambda_{\text{NND}} = 1$).

### H. Forward and backward transfer analysis

Forward transfer (FWT) and backward transfer (BWT) are commonly used in continual learning literature [32,38]. The former measures the capacity of the model to generalize to future tasks, whereas the latter shows the capacity of the model to retain the previously acquired knowledge. Specifically, they are defined as follows. Let $T$ again be the total number of tasks for the continual learning stages, we therefore can divide the test set into $T$ segments, each one representing one task. After each task $t$, the model is evaluated with respect to all $T$ test sets. Consequently, we obtain a matrix $R \in \mathbb{R}^{T \times T}$, where the element $R_{i,j}$ is the test performance on task $j$ with the model on task $i$. We use classification accuracy as our evaluation metrics. In addition, we define the random estimation as $r_j$, which represents the test performance on task $j$ using a model with only random initialization. We can define the FWT and BWT as:

$$FWT = \frac{1}{T-1} \left( \sum_{i=2}^{T} R_{i-1,i} - r_i \right).$$  \hspace{1cm} (8)

$$BWT = \frac{1}{T-1} \left( \sum_{i=1}^{T-1} R_{T,i} - R_{i,i} \right).$$  \hspace{1cm} (9)

Similarly, we can define the average accuracy (ACC) as:

$$ACC = \frac{1}{T} \left( \sum_{i=1}^{T} R_{T,i} \right).$$  \hspace{1cm} (10)

It should be noticed that computing the backward transfer for the first task or the forward transfer for the last task have little utility and are excluded from Eq. 8 and Eq. 9.

We report the results in Tab. 10 a comparison of our proposed components. Note that PAWS diverges in this setting, leading to a low FWT. Instead, PAWS is better than CSL and NNCSL if we look at BWT alone. It is simply because $R_{T,i}$ and $R_{i,i}$ are all low after the divergence, having not much room for the model to forget. That is, a model cannot forget if it does not learn anything first. This observation confirms the limitation of BWT, as it only shows a relative difference with respect to its own performance, i.e., Eq. 9. Thus, BWT is more suitable to be an additional indicator when the average accuracy of the two methods is close to each other, e.g., NNCSL vs. CSL. Comparing NNCSL and CSL, we notice that the NND helps slightly improve the BWT. What is more interesting is that NND significantly improves FWT. We believe it is because NND stabilizes the representation learning, allowing the model to generalize better to future tasks.

We also notice that the absolute value of BWT is high for both NNCSL and CSL. We suggest that it is because the first task suffers the most from forgetting, as it is trained with a simple task and without any regularization of distillation, but it goes through all continual stages. To verify this assumption, we compute the BWT without the first task: $-11.3$ for NNCSL and $-9.23$ for CSL, which are significantly improved from the BWT scores in Tab. 10.

### I. Visualization of the features

We use t-SNE [49] to project the learned features into a lower-dimensional space and visualize them to qualitatively verify the effectiveness of our proposed method. We apply t-SNE on the deep features $h_u = h(z_u)$ of unlabeled data and color them in the visualization with their ground-truth label. Ideally, if the features are well learned, one can see
Table 10: Forward transfer (FWT) and backward transfer (BWT) of PAWS, CSL and NNCSL in 20-task ImageNet-100.

<table>
<thead>
<tr>
<th>Method</th>
<th>FWT ↑</th>
<th>BWT ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAWS</td>
<td>1.1</td>
<td>-13.7</td>
</tr>
<tr>
<td>CSL</td>
<td>26.8</td>
<td>-18.25</td>
</tr>
<tr>
<td>NNCSL</td>
<td>31.7</td>
<td>-17.15</td>
</tr>
</tbody>
</table>

To have a more detailed view on the feature space, we select the first two classes as examples and visualize them at different training stages using different methods. Fig. 10 confirms that PAWS leads to a blurry boundary and is prone to severe forgetting due to this effect.

References


[42] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. ICML, 2018. 2


[47] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. NeurIPS, 30, 2017. 2


