Deep Class-Incremental Learning From Decentralized Data

Xiaohan Zhang, Songlin Dong, Jinjie Chen, Qi Tian, Fellow, IEEE, Yihong Gong, Fellow, IEEE, and Xiaopeng Hong, Member, IEEE

Abstract—In this article, we focus on a new and challenging decentralized machine learning paradigm in which there are continuous inflows of data to be addressed and the data are stored in multiple repositories. We initiate the study of data-decentralized class-incremental learning (DCIL) by making the following contributions. First, we formulate the DCIL problem and develop the experimental protocol. Second, we introduce a paradigm to create a basic decentralized counterpart of typical (centralized) CIL approaches, and as a result, establish a benchmark for the DCIL study. Third, we further propose a decentralized composite knowledge incremental distillation (DCID) framework to transfer knowledge from historical models and multiple local sites to the general model continually. DCID consists of three main components, namely, local CIL, collaborated knowledge distillation (KD) among local models, and aggregated KD from local models to the general one. We comprehensively investigate our DCID framework by using a different implementation of the three components. Extensive experimental results demonstrate the effectiveness of our DCID framework. The source code of the baseline methods and the proposed DCIL is available at https://github.com/Vision-Intelligence-and-Robots-Group/DCIL.

Index Terms—Catastrophic forgetting, continuous learning, incremental learning (IL), knowledge distillation (KD).

I. INTRODUCTION

Deep models have achieved great success in a wide range of artificial intelligence research fields [1], [2], [3], [4], [5], [6], [7]. Nevertheless, they have been shown to prone to the catastrophic forgetting problem [8]. Catastrophic forgetting refers to the phenomenon where the performance of the deep model degrades seriously when evolving the model for new data. In response to this urgent problem, incremental learning (IL) [9], [10], [11], [12], [13], [14], [15], a.k.a. continual learning [16], [17], [18], [19], which is targeted at learning continuous incoming data streams while getting away with catastrophic forgetting, has drawn increasing attention.

The current IL framework requires deep neural network (DNN) models to process continuous streams of information in a centralized manner. Despite its success, we argue that such a centralized setting is often impossible or impractical. More and more data emerge from and exist in “isolated islands,” which may be subject to various regularization or requirements in privacy. It is not always allowed to move data and use data out of their owners. In addition, continuous inflows lead to a huge amount of data located in different repositories, which may cause huge communication and computational burden in bringing them together into a single repository for learning.

Therefore, it is crucial to enable learning models to be deployed in scenarios, where data are located in different places and the learning process to be performed across time beyond the bounds of a single repository. Nevertheless, no existing machine learning paradigm, such as the IL and distributed learning (DL) are able to handle such complex scenarios and thus leave us a big challenge, as illustrated by Table I. In IL [9], [10], [11], [12], [13], [20], [21], [22], a model is updated given a data stream continually coming from one single repository. On the contrary, in DL and federated learning (FL) [23], [24], multiple models learned by different repositories are aggregated to a general model. Clearly, IL cannot process data from multiple repositories, while DL and FL cannot provide a mechanism to handle continuous data streams.

In this article, we raise the concern about such a new challenging scenario, where deep IL shall be performed in a decentralized manner, as illustrated in Fig. 1. To meet this challenge, it is required to enable deep learning models to learn from new data residing on local sites like end devices, and in turn, to promote the performance of the general model on the main site continually.

| TABLE I | DIFFERENT NATURE OF DATA SOURCES AMONG TRADITIONAL CENTRALIZED CIL, DL AND FL, AND OUR PROPOSED DCIL |
|---------|----------------------------------------------------------------------------------------------------------------------------------|
| CIL, DL / FL DCIL | continuous data streams | ☑ | ☑ |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Take a smart family photo album as an example, where a considerable number of photographs can be taken on occasion separately by different smartphones. A pretrained CNN model is deployed in a central console as a general model shared by the family members. To process and analyze the inflows of photographs in real-time, a group of local models is deployed and maintained in each smartphone. As the number of photographs may increase rapidly, the local CNN models are required to learn from and adapt to newly emerging photographs, which may distinctly vary from historical ones. Once local models are updated, the general model shall communicate with the local ones, learn from them, and updates itself accordingly, without copying all users’ photographs in a single repository. The decentralized IL algorithm can also be used in the field of multirobot collaboration, such as welcoming robots, that can recognize human identities. Each welcoming robot learns to identify guests over time and learn to recognize new guests without forgetting previous guests even if data on previous guests is unavailable. All welcoming robots can exchange models, rather than data with guest privacy to each other in order to generate the main model that can recognize all guests.

In response to this demand, we study the paradigm of decentralized class-IL (DCIL). In DCIL, it is avoided to upload the data from local sites to the main site. It, thus, becomes challenging to learn a general model with such limited information. To kick off relevant research, we define a rationale for DCIL learning and evaluation protocol on mainstream CIL datasets. Moreover, under the protocol, we develop a DCIL paradigm to transform typical (centralized) CIL approaches to their corresponding decentralized counterparts and built baseline DCIL results.

Along with its promising prospect, there are still several problems with DCIL. On the one hand, as data are distributed among multiple repositories and streams, local sites can only access a portion of the entire dataset, which inherently induces deflected local optimum during data-decentralized learning [24]. On the other hand, directly averaging the local model weights to form a general model may have detrimental effects on the performance of the general model. One reason may lie in the permutation-invariant property of neural network parameters, as one neural net has multiple equivalent counterparts with different orders of the same parameters [25].

As a result, a phenomenon of model drift on the general model may occur, as shown in Fig. 2.

To further solve the challenge in DCIL and the limitations of the basic DCIL paradigm, we propose a novel DCIL framework, termed decentralized composite knowledge incremental distillation (DCID) framework that enables learning knowledge across time and multiple repositories. There are three main steps in the proposed DCID framework. First, we introduce a data-decentralized learning mechanism to perform CIL. Second, we propose a collaborated knowledge distillation (KD) method to exchange knowledge among local models so that local models can self-consistently evolve, without the supervision from the general model. Finally, we design an aggregated KD method to transfer knowledge from multiple local models to update the general model. The proposed approach outperforms the baseline DCIL algorithms, with a communication cost at the same level.

Briefly, the contributions of this article are manifold.

1) We recognize the importance and initiate the study of DCIL. Compared with the popularly studied CIL, the problem setting of DCIL is more practical and challenging.
2) We propose a basic DCIL paradigm to decentralize state-of-the-art CIL approaches and provide baseline results for the DCIL study.
3) We propose a decentralized deep CIL framework DCID, which consistently outperforms the baselines under various settings.

II. RELATED WORK

This study is relevant to CIL and DL/FL.

A. Deep Neural Networks

DNNs have shown great ability to represent highly complex functions. DNNs, especially deep convolutional neural networks (CNNs) [3], have yielded breakthroughs in image classification and detection tasks. A large number of deep network structures and training techniques have been proposed. After the success of AlexNet [3], deep residual network (ResNet) [1] has become one of the most groundbreaking networks in the deep learning community in the last few years. By using ResNets, the researcher can train up to hundreds of layers and achieves excellent performance. To address the overfitting issue for training DNNs, the dropout technique [26], [27]
is further proposed to regularize the model parameters by randomly dropping the hidden nodes of DNNs in the training steps to avoid co-adaptations of these nodes. Lately, the nonregularity of data structures has led to recent advancements in graph convolutional networks (GCNs) [28]. The work [29] presents a new mini-batch GCN, which allows to reduce the computational cost of traditional GCNs. Moreover, due to the powerful representation ability with multiple levels of abstraction, deep multimodal representation learning [30] has attracted much attention in recent years. The study [31] provides a general multimodal deep learning framework and a new fusion architecture for geoscience and remote sensing image classification applications.

B. Class-Incremental Learning

Continual learning/IL [16] aims at learning from evolving streams of training data. There are two branches of IL: online IL [32] that the model is single-pass through the data without task boundaries, and offline IL that the model can be trained in offline mode in each incremental session. In this paper, we mainly focus on the latter one. There are two major categories of IL: task IL and CIL. A group of studies work on the task-IL scenario [33], [34], [35], [36], [37], [38], [39], where a multilayer structure is used. On the contrary, the CIL task maintains and updates a unified classification head and thus is more challenging. This article mainly focuses on the CIL approaches.

CIL is targeted at continually learning a unified classifier until all encountered classes can be recognized. To prevent the catastrophic forgetting problem, a group of CIL approaches transfer the knowledge of old classes by preserving a few old class anchor samples into the external memory buffer. Many approaches, such as incremental classifier and representation learning (iCaRL) [20] and end-to-end incremental learning (EEIL) [40] use KD, compute the different types of distillation loss functions. KD is a technique to transfer learned knowledge from a trained neural network (as a teacher model) to a new one (as a student model) [41], [42], [43], [44]. KD for CIL is typically used in centralized settings before deployment in order to reduce the model complexity without weakening the predictive power [20], [22], [40], [45], [46]. Later, studies, such as learning a unified classifier incrementally via rebalancing (LUCIR) [22], BiC [46], and maintaining discrimination and fairness in class incremental learning (MDP) [47], focus on the critical bias problem that causes the classifier’s prediction biased toward the new classes by using cosine distance classifiers or an extra bias-correction layer to fix output bias. More recently, topology-preserving class-incremental learning (TPCIL) [21] puts forward the elastic Hebbian graph and the topology-preserving loss to maintain the topology of the network’s feature space. CER [48] proposes a coordinating experience replay approach to constrain the rehearsal process, which shows superiority under diverse IL settings. Furthermore, to utilize the memory buffer more efficiently, memory efficient class-incremental learning for image classification (MeCIL) [49] proposes to keep more auxiliary low-fidelity anchor samples, rather than the original real-high-fidelity anchor samples. Nearly, [50] recognizes the importance of the global property of the whole anchor set and designs an efficient derivable ranking algorithm for calculating loss functions.

It is worth noting that existing (class) IL approaches are studied in a centralized manner. They can only work in a situation where data keep coming from a single repository. Thus, it cannot handle cases where new data emerge from distributed sources.

C. Distributed Learning and Federated Learning

DL is mainly for training data in parallel scenarios with excessive data efficiency. Both data and workloads are divided into multiple work nodes/sites so that the burden of learning the local data of each working node is within the tolerance. In each site, a local model is trained. Local models then communicate with other work nodes in accordance with certain rules like Parameter Server [51]. The server node receives the local model from different work nodes. To integrate and build a general machine learning model, there are studies by simply averaging the model parameters to obtain a general model, solving a conformance optimization problem such as ADMM [52] and BMUF [53], or by model integration like ensemble learning [54].

FL is a popular distributed framework, which enables the creation of a general model through many local sites. The global model is aggregated by the parameters learned at the local sites on their local data. It involves training models over remote end devices, such as mobile phones. One typical FL method, namely, Federated Averaging (FedAvg) [55] aggregates local parameters with weights proportional to the sizes of data on each client. To reduce the communication costs, sparse ternary compression (STC) [56] and temporally weighted aggregation federated learning (TWAFL) [57] compress both the upstream and downstream communications. To reduce global model drifts, FedProx [23] incorporates a proximal term to restrict local models closer to the global model. FedMA [58] matches individual neurons of the neural networks layerwise before averaging the parameters due to the permutation invariance of neural network parameters. There are also relevant works like FedMeta [59], which combine FL with meta-learning and share a parameterized algorithm (or meta-learner) instead of a global model. FedMAX [60] introduces a prior based on the principle of maximum entropy for accurate FL. Moreover, data-sharing FedAvg [61] uses a public dataset between the server side and local sides. Furthermore, FedDF [62] leverages KD technique to aggregate knowledge from local models to refine a robust global model, and performs parameter averaging as done in FedAvg [55].

Nevertheless, most of the research on DL, including FL, today is still performed solving closed tasks which would hardly lead to a more open-world, long-term problem, where things keep changing over time. There are only a few exceptions which is aiming at incrementally learning over multimodes without aggregating data [63], [64]. However, they are in a very preliminary stage. First, they use linear neural nets, which seriously limit the applications and fail to connect to modern deep IL methods. Second, they only investigate a one-learning-session setting, which is apart from real-continual learning over time. Thus, these studies cannot be applied to complicated decentralized deep IL scenarios.
III. DECENTRALIZED CLASS-INCREMENTAL LEARNING

A. Problem Description

We now define the DCIL problem setting as follows. The training dataset $D = \{(x, y) \mid x \in X, y \in L\}$ consists of images from an image set $X$ and their labels from a predefined common label space $L = \{1, \ldots, C\}$, where $C$ is the total number of classes. $D$ is divided into $T$ independent training sessions $D = \{D^{(1)}, D^{(2)}, \ldots, D^{(T)}\}$, where $D^{(t)} = \{(x^{(t)}, y^{(t)}) \mid x \in X^{(t)}, y \in L^{(t)}\}$. Note that the training sets of different sessions are disjoint, so do the label sets, i.e., $X^{(t)} \cap X^{(p)} = \emptyset$ and $L^{(t)} \cap L^{(p)} = \emptyset$ for $t \neq p$.

The goal of DCIL is to obtain a general model $\theta^{(t)}$, which generalizes well to classify new samples of all seen classes probably appearing in any local sites. The learning phase of each session is as follows. At session $t$, a general model $\theta^{(t-1)}$ is prepared before the training stage of the session starts. The goal of this session is to update $\theta^{(t-1)}$ to a new general model $\theta^{(t)}$ so that its performance on $D^{(t)}$ can be improved. Unlike conventional IL settings where the data of the session $D^{(t)}$ is centralized, in the DCIL setting, $D^{(t)}$ is decentralized and distributed to $M$ data owners (local sites). Let $D^{(t)} = \{D^{(t)}_1, \ldots, D^{(t)}_M\}$, where $D^{(t)}_m = \{(x_m^{(t)}, y_m^{(t)}) \mid x_m \in X^{(t)}_m, y_m \in L^{(t)}\}$, $X^{(t)}_1 \cup X^{(t)}_2 \cup \cdots \cup X^{(t)}_M = X^{(t)}$, and $X^{(t)}_m \cap X^{(t)}_n = \emptyset$ for $m \neq n$. Note that all $M$ data owners share a common class label sets of this session $L^{(t)}$. As the general model is not allowed to access $D^{(t)}$ straightforwardly, $\theta^{(t-1)}$ has to be distributed to each data owner, and thus, there are $M$ copies of $\theta^{(t-1)}$, i.e., $\theta_1^{(t-1)}, \ldots, \theta_M^{(t-1)}$, deployed locally to each data owners when session learning starts. Then they continually learn and update to maximize their performance on each of the $X^{(t)}_m$ separately, without forgetting the knowledge learned in previous sessions. Finally, the learned knowledge embedded in the updated local models $\Theta^{(t)} = \{\theta_1^{(t)}, \ldots, \theta_M^{(t)}\}$ is transmitted to the main site for updating general model $\theta^{(t)}$.

B. DCD

In this section, we propose the DCID framework. As shown in Fig. 3, DCID mainly consists of three steps. First, decentralized incremental knowledge distillation (DID) performs CIL in a data-decentralized setting. Second, decentralized collaborative knowledge distillation (DCD) uses collaborated KD among local models. Third, decentralized knowledge aggregated distillation (DAD) provides an aggregated KD mechanism to update the general model.

1) Decentralized Knowledge Incremental Distillation: In the step of DID, there are two categories of knowledge that the DID learner have to acquire: knowledge from data of the current session (i.e., new-class data) and one from data or models of the historical sessions (i.e., old-class data). First, as constrained by the data sharing policy in the DCIL setting, new data of the session can only be accessed by corresponding local data owners (local sites) and cannot be shared by other sites. As a result, the model of the previous session has to be distributed and then deployed locally in a decentralized manner. Second, to avoid catastrophic forgetting, it is important to transfer knowledge from models of the previous session to the local models of the current session, as affirmed by [20], [65]. KD [41] is a typical model compression and acceleration technique to transfer knowledge from large teacher models to lighter easier-to-deploy student models. This technique, as a regularizer, improves the performance of student models by giving extra soft targets with higher information entropy, rather than the one-hot label (hard targets). Modern CIL methods usually rely on a small anchor set to maintain the memory of historical sessions, which has shown to be an effective mechanism to avoid catastrophic forgetting [16].

To meet these challenges, we provide a paradigm to distribute existing class IL approaches to multiple local sites, where CIL can be performed locally and distributively. More specifically, suppose there are $M$ local sites, the loss function $\ell^{(m)}_I$ for the $m$th local sites in DID is composed of an anchor loss $\ell^{(m)}_a$ and a classification loss for new data $\ell^{(m)}_c$

$$\ell^{(m)}_I = \ell^{(m)}_c + \lambda \ell^{(m)}_a$$  \hspace{1cm} (1)

where $\lambda$ is the parameter controlling the strength of the two losses.

The anchor loss $\ell^{(m)}_a$ regularizes the new local model by minimizing the gap between its behavior and the old local model on the anchor set $A^{(t-1)}_m$ for each data owner. Through $\ell^{(m)}_a$, knowledge from models of the previous session can transfer to the local models of the current session.

Generally speaking, the anchor set is composed of the most representative instances of the local data that local models encountered in previous sessions. We use the herding method [20], [66] to construct the anchor sets for local models. For each newly encountered class $c$, the instances whose
feature vectors are the closest to the average feature vector $\mu_c$ of the class are selected as anchors, iteratively. Given the $k - 1$ selected anchors, the $k$th anchor $a^k_m$ of the current class $c$ is added to the anchor set for representing this class of the local model $[\theta_m]$ as follows:

$$a^k_m = \arg \min_{x \in X^m_c} \left\| \mu_c - \frac{1}{k} \sum_{j=1}^{k-1} \phi(x; [\theta_m]) + \frac{1}{k} \sum_{j=1}^{k-1} \phi(a^j_m; [\theta_m]) \right\| \quad (2)$$

where $X^m_c$ is the instance set of class $c$ of local site $m$ and $\phi(\cdot; \theta)$ is the feature function of a backbone CNN model with parameters $\theta$. Note that the anchor set $A^0_m$ is obtained by adding new anchors from session $t$ to the previous anchor set $A^{m-1}_m$, respectively.

It is worth mentioning that (1) provides a generic way of performing the IL process in local sites. By choosing a certain local incremental learner, $\ell^c_m$ and $\ell^m$ can be determined and $\ell^m$ can be minimized. Thus, a large group of modern anchor-based CIL approaches as mentioned in Section II-A can be optional. In Section IV, we investigate four state-of-the-art CIL methods, including iCARL [20], LUCIR [22], ERDIL [67] and TPICIL [21] to the DCIL setting and show that our proposed DCID performs well regardless of the selection of particular CIL approaches in DID.

2) Decentralized Collaborated Knowledge Distillation: As the local users can only access a small portion of data and are not allowed to reach the images owned by other local sites, the cluster centers estimated locally are inevitably biased. We propose an efficient yet effective KD mechanism, termed DCD.

The key idea of the DCD is making the ensemble of local models as the teacher’s knowledge to guide local model learning, which has richer information entropy than the individual local model [68]. To this end, each local model (as a student), sees the ensemble of the whole local models as a teacher. Consequently, the procedure of DCD is as follows.

First, at the start of the new session $t$, a small shared dataset $S^{(t)}$ is built by collecting a few samples without storing their labels from each local site, respectively. $S^{(t)}$ contains the information that can be shared among local sites and shall be carefully set. When $|S^{(t)}|$ is too large, the burden of communicating will increase, while when $|S^{(t)}|$ is too small, the performance of DCD will be declined. As $S^{(t)}$ is accessible to all local sites, given an instance $x \in S^{(t)}$, the output (before the Softmax layer) of each local model $[\hat{f}([\theta_m^{(t)}], x)]$ can be computed.

Second, for $x \in S^{(t)}$, the ensemble output is computed by linear combination of the output provided by local models $[\hat{y}^{(t)}_m] = f([\theta^{(t)}_m], x)$. We use the weighted average output as a reasonable, high-quality soft target to perform KD. Let $[\hat{y}^{(t)}_m, \omega^{(t)}_m] = \sum_{m=1}^{M} \omega^{(t)}_m [\hat{y}^{(t)}_m], \omega^{(t)}_m$ are positive numbers and $\sum_{m=1}^{M} \omega^{(t)}_m = 1$.

Third, each local model learns the knowledge from the ensemble model. For each local site, the decentralized collaborated KD loss $\ell^c_m$ over the dataset $S^{(t)}$ is minimized

$$\ell^c_m = \sum_{S^{(t)}} \kappa \left( \frac{\exp ([\hat{y}^{(t)}_m]_0 / \tau_1)}{\sum_{n=1}^{M} \exp ([\hat{y}^{(t)}_m]_0 / \tau_1)} \right) \left( \frac{\exp ([\hat{y}^{(t)}_m]_0 / \tau_1)}{\sum_{n=1}^{M} \exp ([\hat{y}^{(t)}_m]_0 / \tau_1)} \right) \quad (3)$$

where $\tau_1$ is the distillation temperature parameter that controls the shape of the distribution for distilling richer knowledge from the ensemble teacher output. $n = |L^{(t)}|$ is the number of new classes in the current session. The Kullback–Leibler divergence is chosen as the fundamental distillation loss function $\kappa$.

After that, each local model $[\theta_m^{(t)}]$ updates to $[\theta_m^{(t+1)}]$. It is worth mentioning that only the several samples without their labels and corresponding output are shared between these local models. This mechanism is without any further supervised signals and it is easy to implement. Moreover, the communication cost during the process of DCD is moderate. Furthermore, data privacy is guaranteed as only a few images are shared, without providing corresponding labels.

3) Decentralized Aggregated Knowledge Distillation: In the final stage, the decentralized aggregated KD method transfers the knowledge of multiple local models back to the main site to update the general model $\theta^{(t)}$. One naive solution is to use the most popular aggregation method in FL, i.e., FedAvg [55] to fulfill this function. Nevertheless, the shared dataset $S^{(t)}$ and the set of ensemble output maintained by the second stage DCD provides extra opportunities to distill knowledge and further improve the aggregation results. As a result, we design an effective two-step DAD solution to refine the general model as follows.

At first, the general model $\hat{\theta}^{(t)}$ is initialized by aggregating all local models using FedAvg [55]. That is, general model $\hat{\theta}^{(t)}$ aggregates local models $[\theta^{(t)}_1, \ldots, \theta^{(t)}_M]$ by weighted averaging

$$\hat{\theta}^{(t)} = \sum_{m=1}^{M} \frac{N^{(t)}_m}{N^{(t)}} \theta^{(t)}_m, \quad (4)$$

where $N^{(t)}$ is the number of training samples over all local sites, and $N^{(t)}_m$ is the number of training samples in the $m$th local site.

Then, the main site distills the ensemble of all the local models (as teachers) into one single-general model (as a student). To this end, we use the same ensemble approach as the DCD process. The local models are evaluated on mini-batches of data from the current shared dataset $S^{(t)}$ and their output are aggregated as teacher output $[\hat{z}^{(t)}_1]$. The student output $[\hat{z}^{(t)}_1] = f([\hat{\theta}^{(t)}], x)$ are generated from the initial general model evaluated on $S^{(t)}$.

To secure an enhanced performance, data on $S^{(t)}$ and the corresponding teacher output and student output are shuffled. We then minimize the decentralized aggregated KD loss $\ell_A$ over the dataset $S^{(t)}$

$$\ell_A = \frac{\sum_{n=1}^{M} \exp ([\hat{z}^{(t)}]_1/ \tau_2)}{\sum_{n=1}^{M} \exp ([\hat{z}^{(t)}]_1/ \tau_2)} \left( \frac{\exp ([\hat{z}^{(t)}]_1/ \tau_2)}{\sum_{n=1}^{M} \exp ([\hat{z}^{(t)}]_1/ \tau_2)} \right) \left( \frac{\exp ([\hat{z}^{(t)}]_1/ \tau_2)}{\sum_{n=1}^{M} \exp ([\hat{z}^{(t)}]_1/ \tau_2)} \right) \quad (5)$$

We use the same form of $\kappa$ as in (3). $\tau_2$ is the distillation temperature parameter. Note that we only transfer corresponding output of $S^{(t)}$ from each local site to the main site, with only a small amount of extra communication costs the same as DCD.
4) Overall Learning Procedure: We recap the operations performed by the local site and the main site at session \( t \) as follows.

Each **local site** uploads the randomly chosen samples without sharing their labels of new classes to the main site and downloads the shared dataset \( S^{(t)} \). Then, the following processes are repeated until convergence. First, each local site updates the general model \( \theta^{(t-1)} \) of the previous session using (1) to obtain \( \hat{\theta}^{(t)}_m \). Second, each local site performs DCD, finetunes its model using the ensemble output according to (3) and obtains \( \hat{\theta}^{(t)}_m \). Third, each local site uploads its model parameter \( \hat{\theta}^{(t)}_m \) and output on \( S^{(t)} \) to the main site.

The **main site** receives the uploaded samples from all local sites and constructs a shared dataset \( S^{(t)} \). Then, the following processes are repeated for \( R \) rounds. Each round represents the process by that the general model to be distributed, and the new general model is obtained after local model training and aggregation. The main site first broadcasts the latest general model \( \theta^{(t-1)} \) to local sites. Second, it calculates the ensemble output by using the uploaded output from local sites and distributes them back to the local sites. Third, the main site aggregates model parameters sent from all the local sites by taking a weighted average of them to initialize the general model parameters \( \hat{\theta}^{(t)} \) and then fine-tunes it by using the ensemble of output from local sites to get \( \theta^{(t)} \). Finally, the main site broadcasts the updated \( \theta^{(t)} \) to all local sites.

The overall learning procedure of the proposed DCID framework in one session is summarized in Algorithm 1. It is worth mentioning that besides the upload and download of model parameters, our DCID only shares a very limited number of training samples without sharing their labels among local sites and the main site, which not only protects data privacy but also minimizes the communication cost.

### C. Baseline Approaches

As we focus on a new learning paradigm, it is desired to provide baseline results and build a benchmark for the DCIL study. To meet this need, we develop a basic decentralized framework to expand typical CIL methods like the **four** introduced in Section III-B to their DCIL counterparts. The overall procedure of the proposed basic DCIL framework is summarized in Algorithm 2 and described as follows.

Given a particular class-incremental learner, at the beginning of session \( t \), the main site distributes the general model \( \theta^{(t-1)} \) to \( M \) local data owners (Step 1). Then, each of the data owners updates the local model using its own training data \( X^m \) to adapt to new classes while performing a certain kind of forgetting alleviation mechanism using a few old-class anchors \( A^{(t-1)}_m \) (Step 4). Note that \( X^{(t)}_m \) and \( A^{(t-1)}_m \) in each local site will not be transmitted to other local sites or the main site. The local anchor set \( A^{(t-1)}_m \) is then updated to be \( A^{(t)}_m \) (Step 5). Finally, the updated local models are transmitted to the main site and the general model evolves as the simple weighted average of all the local models based on the amount of data trained at the current session (Step 7). It is worth mentioning that Algorithm 2 works as well for IL methods without using a historical anchor set, by simply setting \( A = \emptyset \) and removing Step 5.

#### Algorithm 1 DCID Framework

**Input:** Training set \( \{D^{(t)}_1, \ldots, D^{(t)}_M\} \) of the current session, the anchor set \( \{A^{(t-1)}_1, \ldots, A^{(t-1)}_M\} \), the shared dataset \( S^{(t)} \) and the general model \( \theta^{(t-1)} \) after the previous session.

**Output:** The updated general model \( \theta^{(t)} \) of the current session.

1. Copy the general model \( \theta^{(t-1)} \) of the main site and distribute to \( M \) data owners as \( \{\theta^{(t)}_m\} \);
2. for each round \( r = 0, 1, \ldots, R - 1 \) do
3. for each local site \( m = 0, 1, \ldots, M - 1 \) in parallel do
4. Update local models \( \{\theta^{(t)}_m\}_0 \) by minimizing \( \ell^m \);
5. Compute local output \( [z^{(t)}_m]_0 \) on \( S^{(t)} \);
6. Compute \( A^{(t)}_m \) according to Eq.2;
7. end for
8. \( [z^{(t)}]_0 = \sum_{m=1}^M \omega^{(t)}_m [z^{(t)}_m]_0 \);
9. for each client \( m = 0, 1, \ldots, M - 1 \) in parallel do
10. Update local models \( \{\theta^{(t)}_m\}_1 \) by minimizing \( \ell^c \);
11. end for
12. \( \hat{\theta}^{(t)} \leftarrow \sum_{m=1}^M \frac{N^m}{N} \hat{\theta}^{(t)}_m \);
13. for each local site \( m = 0, 1, \ldots, M - 1 \) in parallel do
14. Compute local output \( [z^{(t)}_m]_1 \) on \( S^{(t)} \);
15. end for
16. \( [z^{(t)}]_1 = \sum_{m=1}^M \omega^{(t)}_m [z^{(t)}_m]_1 \);
17. Update general model \( \theta^{(t)} \) by minimizing \( \ell_A \);
18. end for

#### Algorithm 2 Basic DCIL Framework

**Input:** Training set \( \{D^{(t)}_1, \ldots, D^{(t)}_M\} \) of the current session, the anchor set \( \{A^{(t-1)}_1, \ldots, A^{(t-1)}_M\} \), and the general model \( \theta^{(t-1)} \) after the previous session.

**Output:** The updated general model \( \theta^{(t)} \) of the current session.

1. Copy the general model \( \theta^{(t-1)} \) of the main site and distribute to \( M \) data owners as \( \{\theta^{(t)}_m\} \);
2. for each round \( r = 0, 1, \ldots, R - 1 \) do
3. for each local site \( m = 0, 1, \ldots, M - 1 \) in parallel do
4. Update local models \( \{\theta^{(t)}_m\}_0 \) by minimizing \( \ell_{loc} \);
5. Compute \( A^{(t)}_m \) according to Eq. 2;
6. end for
7. \( \theta^{(t)} \leftarrow \sum_{m=1}^M \frac{N^m}{N} \theta^{(t)}_m \).
8. end for

The proposed framework is carefully designed so that popular federated updating and aggregation methods, such as FedAvg, FedMAX [60], and FedProx [23] can be used as a plug-in module to update the local models and gather them as a whole into a general one. The selection of these methods can be easily controlled by defining different \( \ell_1 \) in Step 4. When using FedAvg for aggregation, \( \ell_1 \) is defined as follows and the resulted DCID method is referred to as “DCIL w/FedAvg”:

\[
\ell_1 = \ell^m + \lambda \ell^0
\]
where $\ell_m^i$, $\lambda$, and $\ell_m^m$ are the same with those in (1). Alternatively, the methods “DCIL w/FedMAX” can be defined as

$$
\ell_1 = \ell_m^i + \lambda \ell_m^m + \frac{1}{B} \sum_{i=1}^{B} \kappa (a_m^m)^T \bar{U}
$$

(7)

where $a_m^m$ refers to the activation vector at the input of the last fully connected layer for sample $i$ on the $m$th local site. $U$ stands for a uniform distribution over the activation vectors. $B$ is a mini-batch size of local training data. $\kappa$ is the Kullback-Leibler divergence and $\beta$ is a hyperparameter used to control the scale of the regulation loss. Moreover, $\ell_1$ for “DCIL w/FedProx” becomes

$$
\ell_1 = \ell_m^i + \lambda \ell_m^m + \frac{\mu}{2} \| \theta_m^i - \theta_m^j \|^2
$$

(8)

where $\mu$ is the parameter controlling the scale of the proximal term. Please refer to [60] and [23] for more details of the FedMAX and FedProx methods.

IV. EXPERIMENTS

To facilitate the study of DCIL, we conduct comprehensive experiments under the DCIL setting to provide baseline results of the DCIL frameworks and evaluate the proposed DCID approach.

A. Data and Setup

Experiments are performed in PyTorch under the DCIL setting on two challenging image classification datasets CIFAR100 [69] and subImageNet [20], [22].

CIFAR100 contains 60,000 natural RGB images over 100 classes, including 50,000 training images and 10,000 test images. It is very popular both in IL works [20], [40] and DL/FL [55] works. Each image has a size of $32 \times 32$. We randomly flip images for data augmentation during training.

SubImageNet contains images of 100 classes randomly selected from ImageNet [70]. There are about 130,000 RGB images for training and 5000 RGB images for testing. For data augmentation during training, we randomly flip the image and crop a $224 \times 224$ patch as the network’s input. During testing, we crop a single-center patch of each image for evaluation.

1) Experimental Setting: For each dataset, we randomly choose half of the classes, i.e., 50 classes as the base classes for the base session (session 0), and divide the rest of the classes into $T = 5$ and 10 sessions for IL.

In each session, there are $M$ local sites, where the general model of the previous session is copied, deployed, and updated locally. We randomly sample the dataset of each session under a balanced and independent and identical distribution (IID) for experiments in Sections IV-B–IV-D. Evaluations for non-IID settings are provided in Section IV-E. $M$ is set to 5 as a basic setting for most experiments in Sections IV-B–IV-E. More comprehensive evaluations about different settings of $M$ in a range of [2, 5, 10, 20] are provided as well in Section IV-D.

We choose four representative CIL approaches: iCARL [20], LUCIR [22], ERDIL [67], and TPCIL [21] as the basic incremental learners. iCARL classifies by using the nearest-mean-of-exemplars rule and the traditional KD to transfer the knowledge. LUCIR incorporates three components, i.e., cosine normalization, feature KD, and interclass separation, to alleviate catastrophic forgetting. ERDIL uses an exemplar relation graph to explore the relations information of exemplars from the old classes and leverages the graph-based relational KD to transfer old knowledge for new class learning. TPCIL maintains the topology knowledge space by an elastic Hebbian graph and typology-preserving loss. We decentralize them to multiple local sites in the DID stage to verify the proposed framework. At the start of each session, each local site initializes classification layer parameters for new classes equally before training for better convergence [55].

A series of session accuracy are recorded on the testing sets at the end of each session from session 0 to the last session, respectively. Two measures, i.e., the average accuracy over such a series of accuracy as well as the final accuracy, namely, the accuracy of the last session are reported for evaluating the performance [12], [20], [21], [22], [67], [71]. The details settings of the parameters are reported as follows.

The anchor number is set to 20 for each class of local sites and the number of shared data for each class is set to 20. During KD, the learning rate is set to $10^{-4}$ and the epoch is set to 5. The distillation temperature parameter in (3) and (5) are set to 5. We have evaluated $\mu$ in (8) for “DCIL w/FedProx” in the interval of [0.02, 10] and found that the results are insensitive to the setting. As a result, we set $\mu = 0.2$. Analogously, we set $\beta = 500$ in (7) for “DCIL w/FedMAX”.

Considering the different scales of the two datasets, we follow mainstream CIL study to choose different backbone networks with corresponding settings. On CIFAR100, we choose the popular 32-layer ResNet [1] as the backbone, as in [22]. Initially, we train the base model for 160 epochs using a mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128. As per the recommended settings from the original papers, we set the hyperparameter $\lambda = 15$ in (1) for TPCIL, $\lambda = 100$ for ERDIL, while $\lambda = 5$ for iCARL and LUCIR. The initial learning rate is set to 0.1 and decreased to 0.01 and 0.001 at epochs 80 and 120, respectively. During decentralized deep IL sessions, we choose the local epoch $E = 10$, with a mini-batch size of 128 for each local site. The learning rate is initially set to 0.01 and decreased by ten times at epoch 10. The number of rounds $R$ is set to 10 in both Algorithms 1 and 2. On subImageNet, we follow [22] and use the 18-layer ResNet as the backbone. We set the hyperparameter $\lambda = 100$ in (1) for ERDIL and $\lambda = 10$ for other CIL methods. We train the base model with a mini-batch size of 128 and the initial learning rate is set to 0.1. We decrease the learning rate to 0.01 and 0.001 after epochs 30 and 50, respectively, and stop training at epoch 100. Then, we fine-tune the model on each subsequent and decentralized training set. The learning rate is initially set to 0.1 and decreased by ten times at epoch 10. We choose the local epoch $E = 30$, with a mini-batch size of 128 for each local site. $R$ is set to 3.
Fig. 4. Comparison between DCID and baselines on CIFAR100. The accuracy values reported here are the mean and the standard deviation of three-times test. (a) iCARL, five sessions. (b) LUCIR, five sessions. (c) ERDIL, five sessions. (d) TPCIL, five sessions. (e) iCARL, ten sessions. (f) LUCIR, ten sessions. (g) ERDIL, ten sessions. (h) TPCIL, ten sessions.

B. Comparison Results Under the IID Setting

Under the IID setting, the training data of each new class in session $t$ is partitioned into each local site randomly, similar to McMahan et al. [55]. Figs. 4 and 5 show the comparison results between our proposed DCID and baseline approaches described in Section III-C on two datasets: CIFAR100 and subImageNet. For stable evaluation results, we run the experiments for three times and report the mean results using four representative class-incremental methods: iCARL, LUCIR, ERDIL, and TPCIL as reported. Each curve reports the mean testing accuracy over sessions. To achieve upper bound performances for reference, we retrain the model at each session with a centralized setting of data, which is denoted as “Centralized.” The green curve reports the accuracy achieved by our proposed DCID, while the crimson, blue, pink, and amber curves report the accuracies of “Centralized,” “DCIL w/FedAvg,” “DCIL w/FedMAX,” and “DCIL w/FedProx,” respectively. The main results are summarized as follows.

1) In all experiments with five and ten-session settings on both two datasets, our DCID consistently outperforms baseline results on each incremental session by a large margin, especially on the challenging subImageNet. The superiority of our method becomes more obvious after learning all the sessions, which shows the effectiveness of long-term IL from decentralized data.

2) On CIFAR100, our DCID frameworks with iCARL, LUCIR, ERDIL, and TPCIL achieve average accuracies of 62.78%, 63.77%, 64.05%, and 64.86%, respectively. As a result, DCID outperforms the second-best one by up to 1.28% in terms of the average accuracy with the LUCIR method. After learning all the sessions, DCID further outperforms “DCIL w/FedMAX” by up to 1.88% in terms of the final accuracy with the LUCIR method. Analogously, under the ten-session setting, our DCID also outperforms baseline approaches both in average accuracy and final accuracy.

3) On subImageNet, under the five-session setting, our DCID with iCARL, LUCIR, ERDIL, and TPCIL achieve the average accuracies of 65.23%, 68.01%, 66.99%, and 70.93%, respectively, which are 4.36%, 3.83%, 3.88%, and 4.20% higher than the second-best method, correspondingly. Furthermore, at the last session, our DCID greatly outperforms the second-best method by up to 5.70%, 6.36%, 6.40%, and 5.62%. Under the ten-session setting, our DCID also outperforms the second-best method with iCARL, LUCIR, ERDIL, and TPCIL by 4.35%, 3.77%, 4.14%, and 3.15% in terms of average accuracies, respectively. Moreover, at the end of the entire learning process, our method outperforms the second-best method by 4.59%, 6.10%, 6.08%, and 4.69%, respectively.

C. Ablation Study

We provide ablation studies on subImageNet to investigate each component’s contribution to the final performance gain and prove the effectiveness and generalization ability of DCID. We conduct the experiments using three different centralized CIL approaches: iCARL, LUCIR, and TPCIL.
The experiments are performed on subImageNet under the five-session IL setting with five local sites. We explore the impact of the DCD module in (3) and the DAD module in (5), respectively. Tables II–IV report the comparative results using iCARL, LUCIR, and TPCIL methods, respectively. Our “Baseline” method is “DCIL w/FedAvg,” in other words, removing the DAD and DCD modules from DCID. “Baseline w/DAD” refers to the “Baseline” method with the DAD module. We summarize the results as follows.

1) Both the DCD and the DAD modules improve the performance of the baseline no matter which of the three basic CIL methods is used. Baseline with DCD outperforms the baseline by up to 3.19%, 4.00%, and 3.49% using iCARL, LUCIR, and TPCIL, respectively. Baseline with DAD exceeds the baseline by up to 2.75%, 3.09%, and 2.54%, correspondingly.

2) When combing the two modules with the baseline, DCID achieves the best average accuracy. It exceeds Baseline with DCD by up to 1.18%, 0.80%, and 1.55% using iCARL, LUCIR, and TPCIL, and Baseline with DAD by up to 1.64%, 1.71%, and 2.50%, respectively. All these results show that our proposed method consistently outperforms the baselines, regardless of the (centralized) CIL methods used. The effectiveness of the proposed DCID and its components is demonstrated.

### D. Key Issues of DCID

1) Effect of the Size of Shared Dataset: To investigate the effect brought by the different amounts of the shared dataset, we further evaluate the methods using the different numbers of shared samples per class on a local site. With
why we choose this parameter to carry out our experiments.

| Number of Shared Samples | Encountered Classes | Average | Acc. |
|--------------------------|---------------------|---------|------|
|                          |                     |         |      |
| 0                        | 85.52               | 70.83   | 63.34|
| 2                        | 85.52               | 73.06   | 67.14|
| 5                        | 85.52               | 73.26   | 68.35|
| 10                       | 85.52               | 73.57   | 68.28|
| 20                       | 85.52               | 73.43   | 68.91|
| 30                       | 85.52               | 73.45   | 69.13|

Fig. 6. Comparison of (a) average accuracy and (b) final accuracy between our method DCID and baseline method “DCIL w/FedAvg” with different numbers of local epochs.

the five sessions and five local site settings, we compare the performance comparison using the LUCIR method in Table V. The number of shared samples per class is in the range of \{0, 2, 5, 10, 20, 30\}. Note that the method with “0” shared samples equals the baseline method “DCIL w/FedAvg.”

We can observe that the larger shared dataset can perform better, while it also brings more communication costs and data privacy issues. Moreover, we can see that the test accuracy is prone to be saturated when the number of shared samples per class is larger than 20 in a local site, which is also the reason why we choose this parameter to carry out our experiments.

2) Effect of the Number of Local Epochs: The number of the local epochs \(E\) is a worth-exploring hyperparameter in DL and FL, which affects the computation on local sites and the tradeoff between communication cost and performance [62]. Fig. 6 compares the test accuracy of \(E\) in the range of \{5, 10, 20, 30, 50\} with 300 total training epochs. The experiments use the LUCIR method with the five-session and five local site settings.

We can observe from Fig. 6 that our DCID method consistently outperforms the baseline method (“DCIL w/FedAvg”) on both average accuracy and final accuracy for all numbers of local epochs.

Specifically, we found that the performance of our method improves the average accuracy and final accuracy of the baseline method by 5.49% and 8.17% on average, respectively. Moreover, we observe that the longer the local training period takes, the better our method works. The reason can be summarized as that longer local training leads to a higher quality of the ensemble, and hence, a better distillation result for the models [72]. In contrast, the performance of the baseline method saturates and even degrades after \(E = 30\), which is consistent with the observations in the previous literature [55], [58]. Fortunately, this phenomenon is alleviated in the proposed DCID.

3) Effect of the Number of Anchors: For anchor-based CIL methods, though storing more anchors may be helpful for the performance, it also brings additional external memory overheads. Table VI reports the average accuracy achieved by using different numbers of anchors per class in a local site. It is observed that the test accuracy is prone to be saturated when the number of anchors per class is over 20. Therefore, when the number of anchors reaches 20, the test accuracy and the (external) memory costs reach a good balance. Furthermore, 20 is also the anchor number per class used in the original LUCIR [22] and TPCIL [21] algorithms for similar considerations. Anchors only bring a little extra external memory overhead in general. Take session 1 in subImageNet as an example, the total anchor memory overhead is about 30M, 60M, 120M, 180M, and 240M, when the anchor numbers per class are 5, 10, 20, 30, and 40, respectively.

4) Effect of the Number of Local Sites: We further analyze the influences when our model communicates with the different numbers of local sites. In the experiments, we use LUCIR as the CIL method on the subImageNet dataset with the five-session setting. The number of local sites \(M\) is in the range of \{2, 5, 10, 20\}. The anchor number is fixed at 100 per class. Table VII shows the average accuracy and the final accuracy on the testing set achieved by different settings of local site number \(M\).

We can observe that with the increase of local site number \(M\), both the accuracies of the baseline and our method increase, which is consistent with observations in [58] and [61]. However, we can still see that our DCID achieves significantly better performance than the baseline in all settings consistently, which demonstrates the efficiency of our method. Moreover, with the increase in the number of local sites, our method has a greater improvement than the baseline method. This improvement is because a larger number of local sites enrich the diversity of models better and obtain higher quality ensemble outputs [72].

5) Computational Costs: We further evaluate the memory requirements and the time costs. The memory costs contain external memory costs and the GPU runtime memory costs. The external memory costs in local sites depend on the maximum number of samples \(N_m^{(i)}\) and anchors \(a_m^{(i)}\) used to update local models \(\{\theta_m^{(i)}\}_0\) by
minimizing the $\ell_m^t$ (at Line 4 in Algorithm 1), the number of shared samples $|S^{(i)}|$ used to update local models $[\theta_m^{(i)}]$ by minimizing $\ell_m^t$ (at Line 10 in Algorithm 1) and compute local output (at Line 5 in Algorithm 1), and the number of anchors $a_m \times |L^{(i)}|$ calculated by (2) to represent the $|L^{(i)}|$ classes in session $t$ (at Line 6 in Algorithm 1). We set the batch size of training local sites to $B$, the epochs of training $[\theta_m^{(i)}]_0$ and $[\theta_m^{(i)}]_1$ to $E_0$ and $E_1$, respectively. If the backward propagation costs of one batch size is $U(1)$, then the costs of training $[\theta_m^{(i)}]_0$ is $U((N_m^{(i)} + a_m^{(i)})/B) \times E_0$ and the costs of training $[\theta_m^{(i)}]_1$ is $U(a_m \times |L^{(i)}|/B) \times E_1$. Since there is no backward propagation calculating, the time cost of computing local output (at Line 5 in Algorithm 1) and calculating anchors (at Line 6 in Algorithm 1) is trivial.

In addition, we evaluate the time costs of CIL approaches and our DCIL approach, DCID. As both typical CIL and our proposed DCIL approaches generate a unified model when using the same backbone, the inference time will be the same. As a result, we only have to compare the total training time.

We have conducted experiments on the total training time of four CIL algorithms and our DCID on the CIFAR100 dataset with five local sites in one IL session in Table VIII. All training time cost is calculated from the start of training to the end of the last epoch. We conducted experiments with five local sites in one IL session in Table VIII. All four CIL algorithms and our DCIL approach, DCID. As both typical CIL and our DCIL approach. The value of $\alpha$ is a concentration parameter controlling the degree of non-IID-ness among multiple local sites, while $\alpha$ characterizes a prior class distribution over $L$ classes for non-IID settings in DL and FL, as used in [65], [79], and [80].

Therefore, we follow the Dirichlet distribution $\alpha \sim \text{Dir}(\alpha \mathbf{p})$ to synthesize non-IID training data distributions in our experiments. The value of $\alpha$ is a concentration parameter controlling the degree of non-IID-ness among multiple local sites, while $\mathbf{p}$ characterizes a prior class distribution over $L$ classes in incremental session $t$. With $\alpha \rightarrow 0$, each local site holds training samples from only one random class; with $\alpha \rightarrow \infty$, all local sites have identical distributions to the prior class distribution. The prior class distribution is set to a uniform distribution in our experiments. Therefore, a smaller $\alpha$ indicates higher data heterogeneity. In this work, we use the five-session setting as an example, so each session contains ten classes in both CIFAR100 and subImageNet. To better understand the local data distribution we considered in the experiments, we visualize the effects of adopting different $\alpha$ for the five-session setting of CIFAR100 and subImageNet in Fig. 7.

We conduct experiments on $\alpha = 0.1, 1, 10$ and 100 on subImageNet with LUCIR and CIFAR100 with iCARL. We can observe that our DCID consistently outperforms the baseline method “DCIL w/FedAvg,” both in final accuracies and the local models can be trained simultaneously on the local dataset. Therefore, the proposed DCID framework is efficient.

6) Robustness to Data Variability: It is of great importance that the algorithms for machine learning are robust to potential data perturbations. To quantitatively validate the robustness of our method, inspired by the experiments in [73], we perturb test images by jittering the hue of images and evaluate the performance against chromatic changes. We denote the hue of the original image by $H$ and the parameter of the magnitude of hue shift by $\alpha_h$. The hue of the image after processing is randomly sampled within the interval of $[H - \alpha_h, H + \alpha_h]$. We conducted experiments on subImageNet with the LUCIR method to investigate the performances of the baseline and our DCID with different $\alpha_h$. The results are shown in Table IX.

Though the test accuracy is declined by the jittering of image hue, our DCID still outperforms the baseline method, proving the robustness of our method against chromatic perturbations.

E. Evaluation Under the Non-IID Setting

In data heterogeneously distributed settings, training data of new classes in session $t$ are distributed independently of class labels. The Training data usually follows Dirichlet distribution over $L$-class for non-IID settings in DL and FL, as used in [65], [79], and [80].

| Table VII | Comparison of Average Accuracy and Final Accuracy Using Our Method and Baseline With Different Numbers of Local Sites |
|-----------|----------------------------------------------------------------------------------------------------------|
| Methods   | Number of Local Sites |
|-----------|-------------------------------------------------|
| Average   | Baseline | 67.13 | 63.21 | 62.86 | 61.71 |
| Acc.      | DCID (Ours) | 70.07 | 68.01 | 67.61 | 66.27 |
| Final     | Baseline | 56.32 | 51.14 | 48.92 | 45.46 |
| Acc.      | DCID (Ours) | 61.66 | 58.92 | 56.54 | 53.82 |

| Table VIII | Comparison of Training Time of Typical CIL Method and DCID of One Session on CIFAR100 |
|------------|------------------------------------------|
| Methods    | iCARL | LUCIR | ERDIL | TPCIL |
| Typical CIL | 767s  | 1041s | 1094s | 1283s |
| DCID       | 558s  | 662s  | 745s  | 871s  |
We initiate the study of decentralized deep IL, which handles continuous streams of data coming from different sources. It is distinct from existing studies on (deep) IL and DL. IL can only update a model given a data stream coming from a single repository, while neither DL nor FL can handle continuous data streams. The study of decentralized deep IL is thus significant and challenging. To facilitate this study, we establish a benchmark. We then propose a DCID method, which further outperforms the baseline methods consistently by a large margin under different IID and non-IID settings. In the future, the proposed method will be applied to multirobot systems.

ACKNOWLEDGMENT

The authors are also grateful to Liangfei Zhang and Yu Liu for their comments and discussions.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 770–778.

[2] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 3431–3440.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst. (NIPS), vol. 25, Dec. 2012, pp. 1097–1105.

[4] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards realtime object detection with region proposal networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 28, 2015, pp. 1–9.

[5] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “SphereFace: Deep hypersphere embedding for face recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 212–220.

[6] D. Chang et al., “The devil is in the channels: Mutual-channel loss for fine-grained image classification,” IEEE Trans. Image Process., vol. 29, pp. 4683–4695, 2020.

[7] J. Xie, Z. Ma, D. Chang, G. Zhang, and J. Guo, “GPCA: A probabilistic framework for Gaussian process embedded channel attention,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 11, pp. 8230–8248, Nov. 2022.

[8] M. McCluskey and N. J. Cohen, “Catastrophic interference in connectionist networks: The sequential learning problem,” in Psychology of Learning and Motivation, vol. 24. Amsterdam, The Netherlands: Elsevier, Dec. 1989, pp. 109–165.

[9] G. Cauwenberghs and T. Poggio, “Incremental and decremental support vector machine learning,” in Proc. Adv. Neural Inf. Process. Syst., 2001, pp. 409–415.

[10] I. Kuzborskij, F. Orabona, and B. Caputo, “From N to N+1: Multiclass transfer incremental learning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2013, pp. 3358–3365.

[11] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, “Overcoming catastrophic forgetting by incremental moment matching,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, 2017, pp. 1–11.

[12] X. Tao, X. Hong, X. Chang, S. Dong, X. Wei, and Y. Gong, “Few-shot class-incremental learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 12183–12192.

[13] H. Zhao, X. Qin, S. Su, Y. Fu, Z. Lin, and X. Li, “When video classification meets incremental classes,” in Proc. 29th ACM Int. Conf. Multimedia, Oct. 2021, pp. 880–889.

[14] Y. Liu, S. Parisot, G. Slabaugh, X. Jia, A. Leonardis, and T. Tuytelaars, “More classifiers, less forgetting: A generic multi-classifier paradigm for incremental learning,” in Proc. 16th Eur. Conf. Glasgow, U.K.: Springer, Aug. 2020, pp. 699–716.

[15] H. Zhao, Y. Fu, M. Kang, Q. Tian, F. Wu, and X. Li, “MgSvF: Multi-grained slow vs. fast framework for few-shot class-incremental learning,” 2020, arXiv:2006.15524.

[16] G. I. Parisi, R. Kemker, J. L. Part, C. Kanon, and S. Wermter, “Continual lifelong learning with neural networks: A review,” Neural Netw., vol. 113, pp. 54–71, May 2019.
S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, “Overcoming F.
Zenke, B. Poole, and S. Ganguli,” Continual learning through synaptic G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural C. Bucila, R. Caruana, and A. Niculescu-Mizil, “Model compression,” G. M. Park, S. M. Yoo, and J. H. Kim, “Convolutional neural network S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, “ICaRL: Incremental classifier and representation learning,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 2001–2010. T. Tao, X. Chang, X. Hong, X. Wei, and Y. Gong, “Topology-preserving class-incremental learning,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2020, pp. 254–270. S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin, “Learning a unified classifier incrementally via rebalancing,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 831–839. T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” in Proc. Mach. Learn. Syst., vol. 2, 2020, pp. 429–450. S. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, “SCAFFOLD: Stochastic controlled averaging for on-device federated learning,” in Proc. Int. Conf. Mach. Learn., 2020, p. 1–12. M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenwald, T. N. Hoang, and Y. Khazaeni, “Bayesian nonparametric federated learning of neural networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 7252–7261. G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” 2012, arXiv:1207.0580. J. Xie et al., “Advanced dropout: A model-free methodology for Bayesian dropout optimization,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 9, pp. 4605–4625, Aug. 2022. Q. Li, Z. Han, and X.-M. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” in Proc. 32nd AAAI Conf. Artif. Intell., 2018, pp. 1–8. D. Hong, L. Gao, J. Yao, B. Zhang, A. Plaza, and J. Chansusroot, “Graph convolutional networks for hyperspectral image classification,” IEEE Trans. Geosci. Remote Sens., vol. 59, no. 7, pp. 5966–5978, Jul. 2021. J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, “Multimodal deep learning,” in Proc. ICML, 2011, pp. 1–8. D. Hong et al., “More diverse means better: Multimodal deep learning meets remote-sensing imagery classification,” IEEE Trans. Geosci. Remote Sens., vol. 59, no. 5, pp. 4340–4354, May 2021. Z. Cai, O. Sener, and V. Kolotun, “Online continual learning with natural distribution shifts: An empirical study with visual data,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2021, pp. 8281–8290. A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, “Efficient lifelong learning with A-GEM,” in Proc. Int. Conf. Learn. Represent., 2018, pp. 1–20. C. Wu et al., “Memory replay GANs: Learning to generate new categories without forgetting,” in Proc. Adv. NIPS, 2018, pp. 5962–5972. J. Kirkpatrick et al., “Overcoming catastrophic forgetting in neural networks,” Proc. Nat. Acad. Sci. USA, vol. 114, no. 13, pp. 3521–3526, Mar. 2017. S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, “Overcoming catastrophic forgetting by incremental moment matching,” in Proc. Adv. NIPS, 2017, pp. 4652–4662. G. M. Park, S. M. Yoo, and J. H. Kim, “Convolutional neural network with developmental memory for continual learning,” IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 6, pp. 2691–2705, Jun. 2021. F. M. Castro, M. J. Marín-Jímenez, N. Guil, C. Schmid, and K. Alahari, “End-to-end incremental learning,” in Proc. ECCV, 2018, pp. 233–248. G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” Comput. Sci., vol. 14, no. 7, pp. 38–39, 2015. C. Bucila, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2006, pp. 535–541.
Xiohan Zhang received the B.S. degree in chemical engineering and technology from Xi'an Jiaotong University, Xi'an, China, in 2020, where she is currently pursuing the M.S. degree majoring in cyberspace security. Her current research interests include incremental learning and federated learning.

Songlin Dong received the B.E. degree in automation from Xi'an Jiaotong University, Xi'an, China, in 2019, where he is currently pursuing the Ph.D. degree. His research interests include classification, incremental learning, and object detection.

Jinjie Chen received the B.E. degree in software engineering from Xi'an Jiaotong University, Xi'an, China, in 2021, where he is currently pursuing the M.E. degree. His research interests include neural networks and object detection.

Qi Tian (Fellow, IEEE) received the B.E. degree in electronic engineering from Tsinghua University, Beijing, China, in 1992, the M.S. degree in electrical and computer engineering (ECE) from Drexel University, Philadelphia, PA, USA, in 1996, and the Ph.D. degree in ECE from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2002. He was a Full Professor with the Department of Computer Science, The University of Texas at San Antonio (UTSA), San Antonio, TX, USA, from 2002 to 2019. From 2008 to 2009, he was on Faculty Leave with Microsoft Research Asia, Beijing, for one year. From 2018 to 2020, he was the Chief Scientist in computer vision with the Huawei Noah’s Ark Laboratory, Shenzhen, China. He is currently the Chief Scientist in artificial intelligence with Cloud BU, Huawei, Shenzhen. He has authored or coauthored more than 650 refereed journals and conference papers. He has coauthored best papers, including the IEEE International Conference on Multimedia and Expo (ICME) 2019, the ACM Conference on Information and Knowledge Management (CIKM) 2018, the Annual ACM International Conference on Multimedia Retrieval (ICMR) 2015, Pacific Rim Conference on Multimedia (PCM) 2013, International Conference on MultiMedia Modeling (MMM) 2013, ACM Internet Multimedia Computing and Service (ICMCS) 2012, the Top 10 Paper Award in Multimedia Signal Processing (MMSP) 2011, the Student Contest Paper in the International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2006, and the Best Paper/Student Paper Candidate in PCM 2007, ICME 2015, and the ACM Multimedia 2019. His research interests include computer vision, multimedia information retrieval and machine learning.

Dr. Tian received the 2017 UTSA President’s Distinguished Award for Research Achievement, the 2016 UTSA Innovation Award, the 2014 Research Achievement Award from the College of Science, UTSA, the 2010 Google Faculty Award, and the 2010 ACM Service Award. He is an Associate Editor of the IEEE TRANSACTIONS ON MULTIMEDIA (TMM), IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY (TCSVT), ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), and Multimedia Systems (MMS), and on the Editorial Board of the Journal of Multimedia and the Journal of Machine Vision and Applications (MVA). He is also the Guest Editor of IEEE TMM and the Journal of Computer Vision and Image Understanding (CVIU). His Google citation is over 42,600 with an h-index of 96.

Yihong Gong (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical and electronic engineering from The University of Tokyo, Tokyo, Japan, in 1987, 1989, and 1992, respectively.

He was an Assistant Professor with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, for four years. From 1996 to 1998, he was a Project Scientist with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. In 1999, he joined NEC Laboratories America, Princeton, NJ, USA, and established the Media Analytics Group for the laboratories, where he became the Site Manager to lead the entire branch at Cupertino, CA, USA. In 2012, he joined Xi’an Jiaotong University, Xi’an, China, and became a Distinguished Professor of the National Talent Program, the Vice Director of the National Engineering Laboratory for Visual Information Processing, and the Chief Scientist of the China National Key Basic Research Project (973 Project). His current research interests include pattern recognition, machine learning, and multimedia content analysis.

Xiaopeng Hong (Member, IEEE) received the Ph.D. degree from Harbin Institute of Technology (HIT), Harbin, China, in 2010. He is currently a Professor with HIT. He was a Distinguished Research Fellow with Xi'an Jiaotong University, Xi'an, China, and an Adjunct Professor with the University of Oulu, Oulu, Finland. He has authored over 50 articles in journals and conferences, such as the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE (T-PAMI), the IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR), The International Conference on Computer Vision (ICCV), and the AAAI Conference on Artificial Intelligence (AAAI). His current research interests include incremental learning, visual surveillance, and affective computing.

Dr. Hong’s studies about subtle facial movement analysis have been reported by international media, such as the MIT Technology Review and received the 2020 IEEE Finland Section Best Student Conference Paper.