FEDERATED SEMI-SUPERVISED LEARNING WITH INTER-CLIENT CONSISTENCY & DISJOINT LEARNING

Wonyong Jeong\textsuperscript{1}, Jaehong Yoon\textsuperscript{2}, Eunho Yang\textsuperscript{1,3}, and Sung Ju Hwang\textsuperscript{1,3}
Graduate School of AI\textsuperscript{1}, KAIST, Seoul, South Korea
School of Computing\textsuperscript{2}, KAIST, Daejeon, South Korea
AITRICS \textsuperscript{3}, Seoul, South Korea
\{wyjeong, jaehong.yoon, eunhoy, sjhwang82\}@kaist.ac.kr

ABSTRACT

While existing federated learning approaches mostly require that clients have fully-labeled data to train on, in realistic settings, data obtained at the client-side often comes without any accompanying labels. Such deficiency of labels may result from either high labeling cost, or difficulty of annotation due to the requirement of expert knowledge. Thus the private data at each client may be either partly labeled, or completely unlabeled with labeled data being available only at the server, which leads us to a new practical federated learning problem, namely Federated Semi-Supervised Learning (FSSL). In this work, we study two essential scenarios of FSSL based on the location of the labeled data. The first scenario considers a conventional case where clients have both labeled and unlabeled data (labels-at-client), and the second scenario considers a more challenging case, where the labeled data is only available at the server (labels-at-server). We then propose a novel method to tackle the problems, which we refer to as Federated Matching (FedMatch). FedMatch improves upon naive combinations of federated learning and semi-supervised learning approaches with a new inter-client consistency loss and decomposition of the parameters for disjoint learning on labeled and unlabeled data. Through extensive experimental validation of our method in the two different scenarios, we show that our method outperforms both local semi-supervised learning and baselines which naively combine federated learning with semi-supervised learning. The code is available at https://github.com/wyjeong/FedMatch.

1 INTRODUCTION

Federated Learning (FL) (McMahan et al., 2017; Zhao et al., 2018; Li et al., 2018; Chen et al., 2019a;b), in which multiple clients collaboratively learn a global model via coordinated communication, has been an active topic of research over the past few years. The most distinctive difference of federated learning from distributed learning is that the data is only privately accessible at each local client, without inter-client data sharing. Such decentralized learning brings us numerous advantages in addressing real-world issues such as data privacy, security, and access rights. For example, for on-device learning of mobile devices, the service provider may not directly access local data since they may contain privacy-sensitive information. In healthcare domains, the hospitals may want to improve their clinical diagnosis systems without sharing the patient records.

Existing federated learning approaches (McMahan et al., 2017; Wang et al., 2020; Li et al., 2018) handle these problems by aggregating the locally learned model parameters. A common limitation is that they only consider supervised learning settings, where the local private data is fully labeled. Yet, the assumption that all of the data examples may include sophisticate annotations is not realistic for real-world applications. Suppose that we perform on-device federated learning, the users may not want to spend their time and efforts in annotating the data, and the participation rate across the users may largely differ. Even in the case of enthusiastic users may not be able to fully label all the data in the device, which will leave the majority of the data as unlabeled (See Figure 1 (a)). Moreover, in some scenarios, the users may not have sufficient expertise to correctly label the data. For instance, suppose that we have a workout app that automatically evaluates and corrects one’s body posture. In this case, the end users may not be able to evaluate his/her own body posture at all (See Figure 1 (b)).
This leads us to practical problems of federated learning with deficiency of labels, namely Federated Semi-Supervised Learning (FSSL). A naive solution to these scenarios is to simply perform Semi-Supervised Learning (SSL) using any off-the-shelf methods (e.g. FixMatch (Sohn et al., 2020), UDA (Xie et al., 2019)), while using federated learning algorithms to aggregate the learned weights. Yet, this does not fully exploit the knowledge of the multiple models trained on heterogeneous data distributions. To address this problem, we present a novel framework which we refer to as Federated Matching (FedMatch), which enforces the consistency between the predictions made across multiple models. Further, conventional semi-supervised learning approaches are not applicable for scenarios where labeled data is only available at the server (Figure 1 (b)), which is a unique SSL setting for federated learning. Also, even when the labeled data is available at the client (Figure 1 (a)), learning from the unlabeled data may lead to forgetting of what the model learned from the labeled data. To tackle these issues, we decompose the model parameters into two, a dense parameter for supervised and a sparse parameter for unsupervised learning. This sparse additive parameter decomposition ensures that training on labeled and unlabeled data are effectively separable, thus minimizing interference between the two tasks. We further reduce the communication costs with both the decomposed parameters by sending only the difference of the parameters across the communication rounds. We validate FedMatch on both scenarios (Figure 1 (a) and (b)) and show that our models significantly outperform baselines, including a naive combination of federated learning with semi-supervised learning, on the training data which are both non-i.i.d. and i.i.d. data. The main contributions of this work are as follows:

- We introduce a practical problem of federated learning with deficiency of supervision, namely Federated Semi-Supervised Learning (FSSL), and study two different scenarios, where the local data is partly labeled (Labels-at-Client) or completely unlabeled (Labels-at-Server).
- We propose a novel method, Federated Matching (FedMatch), which learns inter-client consistency between multiple clients, and decomposes model parameters to reduce both interference between supervised and unsupervised tasks, and communication cost.
- We show that our method, FedMatch, significantly outperforms both local SSL and the naive combination of FL with SSL algorithms under the conventional labels-at-client and the novel labels-at-server scenario, across multiple clients with both non-i.i.d. and i.i.d. data.

2 Problem Definition

We begin with formal definition of Federated Learning (FL) and Semi-Supervised Learning (SSL). Then, we define Federated Semi-Supervised Learning (FSSL) and introduce two essential scenarios.

2.1 Preliminaries

**Federated Learning** Federated Learning (FL) aims to collaboratively learn a global model via coordinated communication with multiple clients. Let $G$ be a global model and $L = \{l_k\}_{k=1}^K$ be a set of local models for $K$ clients. Let $D = \{x_i, y_i\}_{i=1}^N$ be a given dataset, where $x_i$ is an arbitrary training instance with a corresponding one-hot label $y_i \in \{1, \ldots, C\}$ for the $C$-way multi-class classification problem and $N$ is the number of instances. $D$ is composed of $K$ sub-datasets $D_l = \{x_i^l, y_i^l\}_{i=1}^{N_l}$ privately collected at each client or local model $l_k$. At each communication round $r$, $G$ first randomly selects $A$ local models that are available for training $L' \subset L$ and $|L'| = A$. The global model $G$ then initializes $L'$ with global weights $\theta^G$, and the active local models $l_a \in L'$ perform supervised learning.
We now describe our Federated Matching (FedMatch) where \( \pi \) would be available at the next round \( r + 1 \), and repeat the learning procedure until the final round \( R \).

**Semi-Supervised Learning** Semi-supervised learning (SSL) refers to the problem of learning with partially labeled data, where the ratio of unlabeled data is usually much larger than that of the labeled data (e.g. 1 : 10). For SSL, \( \mathcal{D} \) is further split into labeled and unlabeled data. Let \( \mathcal{S} = \{ x_i, y_i \}_{i=1}^{S} \) be a set of \( S \) labeled data instances and \( \mathcal{U} = \{ u_i \}_{i=1}^{U} \) be a set of \( U \) unlabeled samples without corresponding label. Here, in general, \( |\mathcal{S}| \ll |\mathcal{U}| \). With these two datasets, \( \mathcal{S} \) and \( \mathcal{U} \), we now perform semi-supervised learning. Let \( p_\theta(y|x) \) be a neural network that is parameterized by weights \( \theta \) and predicts softmax outputs \( \hat{y} \) with given input \( x \). Our objective is to minimize loss function \( \ell_{\text{final}}(\theta) = \ell_s(\theta) + \ell_u(\theta) \), where \( \ell_s(\theta) \) is loss term for supervised learning on \( \mathcal{S} \) and \( \ell_u(\theta) \) is loss term for unsupervised learning on \( \mathcal{U} \).

### 2.2 Federated Semi-Supervised Learning

Now we further describe a practical problem of deficiency of labels in federated learning, which we refer to as Federated Semi-Supervised Learning (FSSL), in which the data obtained at the clients may or may not come with accompanying labels. Given a dataset \( \mathcal{D} = \{ x_i, y_i \}_{i=1}^{N} \), \( \mathcal{D} \) is split into a labeled set \( \mathcal{S} = \{ x_i, y_i \}_{i=1}^{S} \) and an unlabeled set \( \mathcal{U} = \{ u_i \}_{i=1}^{U} \) as in the standard semi-supervised learning. Under the federated learning framework, we have a global model \( G \) and a set of local models \( \mathcal{L} \) where the unlabeled dataset \( \mathcal{U} \) is privately spread over \( K \) clients hence \( \mathcal{U}^k = \{ u_i^k \}_{i=1}^{U_k} \). For a labeled set \( \mathcal{S} \), on the other hand, we consider two different scenarios depending on the availability of labeled data at clients, namely the *Labels-at-Client* and the *Labels-at-Server* scenario, of which problem settings and learning procedures will be discussed later.

### 3 Federated Matching

We now describe our *Federated Matching (FedMatch)* algorithm to tackle the federated semi-supervised learning problem. We describe its core components in detail in the following subsections.

#### 3.1 Inter-Client Consistency Loss

Consistency regularization (Xie et al., 2019; Sohn et al., 2020; Berthelot et al., 2019b; a) is one of most popular approaches to learn from unlabeled examples in a semi-supervised learning setting. Conventional consistency-regularization methods enforce the predictions from the augmented examples and original (or weakly augmented) instances to output the same class label, \( ||p_\theta(y|u) - p_\theta(y|\pi(u))|| \), where \( \pi(\cdot) \) is a stochastic transformation function. Based on the assumption that class semantics are unaffected by small input perturbations, these methods basically ensures consistency of the prediction across the multiple perturbations of *same input*. For our federated semi-supervised learning method, we additionally propose a novel consistency loss that regularizes the *models* learned at multiple clients to output the same prediction. This novel consistency loss for FSSL, *inter-client consistency* loss, is defined as follows:

\[
\frac{1}{H} \sum_{j=1}^{H} \text{KL}[p_{\theta^j}^*(y|u)||p_{\theta^j}(y|u)]
\]

where \( p_{\theta^j}^*(y|x) \) are helper agents that are selected from the server based on model similarity to the client (which we describe later), that are not trained at the client (\( * \) denotes that we freeze the parameters). The server selects and broadcasts \( H \) helper agents at each communication round. We also use data-level consistency regularization at each local client similarly to FixMatch (Sohn et al., 2020). Our final consistency regularization term \( \Phi(\cdot) \) can be written as follows:

\[
\Phi(\cdot) = \text{CrossEntropy}(\hat{y}, p_{\theta^j}(y|\pi(u))) + \frac{1}{H} \sum_{j=1}^{H} \text{KL}[p_{\theta^j}^*(y|u)||p_{\theta^j}(y|u)]
\]

where \( \pi(u) \) performs RandAugment (Cubuk et al., 2019) on unlabeled instance \( u \). \( \hat{y} \) is our novel pseudo-labeling technique, which we refer to as the *agreement-based* pseudo label, defined as follows:
where \( \lambda \) are hyper-parameters to control the learning ratio between the terms. We additionally add \( L_2^\theta \) and \( L_1 \)-Regularization on \( \psi \) such that \( \psi \) is sparse, while not drifting far from knowledge that \( \sigma \) has learned. To sum up, our decomposition technique allows us:

- **Preservation Reliable Knowledge from Labeled Data**: We empirically find that learning on both labeled and unlabeled data with a single set of parameters may result in the model to forget what it learned from the labeled data (see Figure 6 (c)). Our method can effectively prevent inter-task interference via utilizing disjoint parameters only for supervised learning.
- **Reduction of Communication Costs**: Sparsity on the unsupervised parameter \( \psi \) allows to reduce communication cost. In addition, we further minimize the cost by subtracting the learned knowledge for each parameter, such that \( \Delta \sigma = \sigma^l - \sigma^u \) and \( \Delta \psi = \psi^l - \psi^u \), and transmit only the differences \( \Delta \psi \) and \( \Delta \sigma \) as sparse matrices for both client-to-server and server-to-client costs.
- **Disjoint Learning**: In federated semi-supervised learning, labeled data can be located at either client or server, which requires the model’s learning procedure to be flexible. Our decomposition technique allows the model for the supervised training to be done separately elsewhere.
work with unlabeled data as described in Figure 1 (b). This is a common case of real-world applications where labeling requires expert knowledge (e.g. annotating medical images, evaluating body postures for exercises), but the data cannot be shared due to privacy concerns. In this scenario, the overall learning procedure is the same as that of conventional federated learning (global model $G$ aggregates updates from the selected subset of clients and broadcasts them), except that active local models $l_{1:A}$ perform semi-supervised learning by minimizing the loss $\ell_{\text{final}}(\theta^*) = \ell_u(\theta^*) + \ell_a(\theta^*)$ respectively on $S^u$ and $U^u$.

**FedMatch Algorithm for Labels-at-Client Scenario** Now we introduce our FedMatch algorithm for the labels-at-client scenario. As shown in Figure 3, which illustrates an example case of the labels-at-client scenario, active local models $l_{1:A}$ at the current round $r$ learn both $\sigma^{r+1}$ and $\psi^{r+1}$ on both the labeled data $U^{l_{1:A}}$ and unlabeled data $U^{u_{1:A}}$ at each local environment. After the completion of local training, the clients update both their learned knowledge $\sigma^{r+1}$ and $\psi^{r+1}$ to the server. The server then aggregates $\sigma^{r+1}$ and $\psi^{r+1}$, respectively, after embedding local models based on model similarity as well as create KD-Tree to rapidly retrieve the top-$H$ nearest neighbors $\psi^{l_{1:H}}$ for each client. At the next round, the server transmits the aggregated $\sigma^{r+1}$ and $\psi^{r+1}$. For helper agents, server retrieves $H$ helper agents, $\psi^{h_{1:H}}$, to each client for every 10 rounds. More details of the training procedures for FedMatch, for the labels-at-client scenario, is described in Algorithm 1.
We now experimentally validate our method, FedMatch. The setting of this task is mostly the same with the Batch-IID task, except we arbitrarily control tasks.  

1) Batch-IID: We use CIFAR-10 for this task and split 60,000 instances into training (54,000), valid (3,000), and test (3,000) sets. We extract 5 labeled instances per class (C=10) for each client (K=100) as labeled set S, and the rest of instances (49,000) are used as unlabeled data U, so that we can evenly split S and U into $S^{1:100}$ and $U^{1:100}$, such that local models $l_t^{1:100}$ learn on corresponding labeled and unlabeled data during training.  

2) Batch-NonIID (class-imbalanced): The setting of this task is mostly the same with the Batch-IID task, except we arbitrarily control the distribution of the number of instances per class for each client to simulate class-imbalanced environments.  

3) Streaming-NonIID (class-imbalanced): In this task, data streams into each client from class-imbalanced distributions. We use Fashion-MNIST dataset for this task, and split 70,000 instances into training (63,000), valid (3,500), and test (3,500) sets. From train set, we extract 5 labeled instances per class (C=5) for each client (K=10) for a labeled set S. We discard labels for the rest of instances to construct an unlabeled set U (62,000). Then, we split S and U into $S^{1:100}$ and $U^{1:100}$ based on a class-imbalanced distribution. For individual local unlabeled data $U_t^{k}$, we again split all instances into $U_t^{k}$, $t \in \{1, 2, \ldots, T\}$, where $T$ is the number of total streaming steps (we set $T=10$). We train each streaming step for 10 rounds. We describe above tasks under Labels-at-Server scenario, $S$ is simply located at server without any partition. Please see Figure 7 in Appendix, which we visualize the concepts of dataset configuration.

---

Algorithm 2 Labels-at-Server Scenario

1: RunServer()  
2: \( \sigma^0, \psi^0 \) \( \sigma^0, \psi^0 \)  
3: for each round \( r = 1, 2, \ldots, R \) do  
4: for each client \( l^r_a \in L^r \) in parallel do  
5: for each local epoch \( e \) from 1 to \( E_c \) do  
6: \( \theta_{\sigma^e+\psi^e} \leftarrow \theta_{\sigma^e+\psi^e} - \eta \nabla l_e(\theta_{\sigma^e+\psi^e}; s) \)  
7: end for  
8: end for  
9: \( L^r \leftarrow \) (select random \( A \) clients from \( L \))  
10: for each client \( l^r_a \in L^r \) do  
11: \( \psi^1_{l^r_a, H} \leftarrow \text{GetNearestNeighbors}(\psi^r) \)  
12: \( \psi^r_a \leftarrow \text{RunClient}(\sigma^{r+1}, \psi^r, \psi^r_{l^r_a, H}) \)  
13: EmbedLocalModel(\( \sigma^{r+1}, \psi^r_a \))  
14: end for  
15: \( \psi^{r+1} \leftarrow \frac{1}{A} \sum_{a=1}^{A} \psi^r_a \)  
16: end for  
17: RunClient(\( \sigma^r, \psi^r, \psi^r_{l^r_a} \) \)  
18: \( \sigma^{r+1} \leftarrow \sigma^r + \psi^r_{l^r_a, H} \) \( \sigma^{r+1} \leftarrow \sigma^r + \psi^r_{l^r_a, H} \)  
19: for each local epoch \( e \) from 1 to \( E_c \) do  
20: for minibatch \( u \in U_{l^r_a} \) do  
21: \( \theta_{\sigma^e+\psi^e} \leftarrow \theta_{\sigma^e+\psi^e} - \eta \nabla l_u(\theta_{\sigma^e+\psi^e}; \theta_{l^r_a, H}, u) \)  
22: end for  
23: end for

FedMatch Algorithms for Labels-at-Server Scenario  
We now describe our FedMatch algorithm for the labels-at-server scenario. As depicted in Figure 4, which describes an illustrative running example for labels-at-server scenario, the global model $G$ learns $\sigma$ on labeled data $S_t^r$ at the server and the active local clients $l_t^r:A$ at the current round $r$ learn $\psi^{r+1:A}$ on unlabeled data $U_t^r:A$ at each local environment. After the completion of local training, clients update their learned knowledge $\psi^{r+1:A}$ to the server. The server then embedding local models based on model similarity and create a KD-Tree for rapid nearest neighbor search for the top-$H$ most similar $\psi^{r+1:A}$ models for each client. At the next round, server transmits its learned $\sigma^{r+1}$ and the aggregated $\psi^{r+1}$ to each client for every 10 communication rounds. Further training details of FedMatch for the labels-at-server scenario is described in Algorithm 2.

6 EXPERIMENTS

We now experimentally validate our method, FedMatch, on three tasks, such as Batch-IID, Batch-NonIID, and Streaming-NonIID, under both scenarios, Labels-at-Client and Labels-at-Server.

6.1 EXPERIMENTAL SETUP

Tasks  
1) Batch-IID: We use CIFAR-10 for this task and split 60,000 instances into training (54,000), valid (3,000), and test (3,000) sets. We extract 5 labeled instances per class (C=10) for each client (K=100) as labeled set S, and the rest of instances (49,000) are used as unlabeled data U, so that we can evenly split S and U into $S^{1:100}$ and $U^{1:100}$, such that local models $l_t^{1:100}$ learn on corresponding labeled and unlabeled data during training.  

2) Batch-NonIID (class-imbalanced): The setting of this task is mostly the same with the Batch-IID task, except we arbitrarily control the distribution of the number of instances per class for each client to simulate class-imbalanced environments.  

3) Streaming-NonIID (class-imbalanced): In this task, data streams into each client from class-imbalanced distributions. We use Fashion-MNIST dataset for this task, and split 70,000 instances into training (63,000), valid (3,500), and test (3,500) sets. From train set, we extract 5 labeled instances per class (C=5) for each client (K=10) for a labeled set S. We discard labels for the rest of instances to construct an unlabeled set U (62,000). Then, we split S and U into $S^{1:100}$ and $U^{1:100}$ based on a class-imbalanced distribution. For individual local unlabeled data $U_t^k$, we again split all instances into $U_t^k$, $t \in \{1, 2, \ldots, T\}$, where $T$ is the number of total streaming steps (we set $T=10$). We train each streaming step for 10 rounds. We describe above tasks under Labels-at-Client scenario. For Labels-at-Server scenario, S is simply located at server without any partition. Please see Figure 7 in Appendix, which we visualize the concepts of dataset configuration.
Table 1: Performance Comparison on Batch-IID & NonIID Tasks We use 100 clients ($F=0.05$) for 200 rounds. We measure global model accuracy and averaged communication costs. Note that the SL (Supervised Learning) models learn on both $S$ and $U$ with full labels, and are utilized as the upper bounds for each experiment.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Labels-at-Client Scenario</th>
<th>Labels-at-Server Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc. (%)</td>
<td>S2C Cost</td>
</tr>
<tr>
<td>FedAvg-SL</td>
<td>58.60 ± 0.42</td>
<td>100%</td>
</tr>
<tr>
<td>FedProx-SL</td>
<td>59.30 ± 0.31</td>
<td>100%</td>
</tr>
<tr>
<td>FedAvg-DDA</td>
<td>46.35 ± 0.29</td>
<td>100%</td>
</tr>
<tr>
<td>FedProx-UDA</td>
<td>47.45 ± 0.21</td>
<td>100%</td>
</tr>
<tr>
<td>FedAvg-FixMatch</td>
<td>47.01 ± 0.43</td>
<td>100%</td>
</tr>
<tr>
<td>FedProx-FixMatch</td>
<td>47.20 ± 0.12</td>
<td>100%</td>
</tr>
<tr>
<td>FedMatch (Ours)</td>
<td>55.13 ± 0.34</td>
<td>97%</td>
</tr>
</tbody>
</table>

CIFAR-10, Batch-IID Task with 100 Clients ($K=100$, $F=0.05$, $H=2$)

<table>
<thead>
<tr>
<th>Methods</th>
<th>Labels-at-Client Scenario</th>
<th>Labels-at-Server Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc. (%)</td>
<td>S2C Cost</td>
</tr>
<tr>
<td>FedAvg-SL</td>
<td>55.15 ± 0.21</td>
<td>100%</td>
</tr>
<tr>
<td>FedProx-SL</td>
<td>57.75 ± 0.18</td>
<td>100%</td>
</tr>
<tr>
<td>FedAvg-DDA</td>
<td>44.35 ± 0.39</td>
<td>100%</td>
</tr>
<tr>
<td>FedProx-UDA</td>
<td>46.31 ± 0.63</td>
<td>100%</td>
</tr>
<tr>
<td>FedAvg-FixMatch</td>
<td>46.20 ± 0.52</td>
<td>100%</td>
</tr>
<tr>
<td>FedProx-FixMatch</td>
<td>45.55 ± 0.60</td>
<td>100%</td>
</tr>
<tr>
<td>FedMatch (Ours)</td>
<td>32.28 ± 0.84</td>
<td>85%</td>
</tr>
</tbody>
</table>

Figure 5: Test Accuracy Curves on Batch-IID & NonIID Tasks We visualize test accuracy curves of model performance corresponding to the Table 1. Note that the SL (Supervised Learning) models learn on both $S$ and $U$ with full labels, and are utilized as the upper bounds for each experiment.

Baselines and Training Details Our baselines are: 1) Local-SL: local supervised learning (SL) with full labels ($S + U$) without sharing locally learned knowledge. 2) Local-UDA and 3) Local-FixMatch: local semi-supervised learning, including UDA and FixMatch, without sharing local knowledge. 4) FedAVG-SL and 5) FedProx-SL: supervised learning with full labels ($S + U$) while sharing local knowledge via FedAvg and FedProx frameworks. 6) FedAvg-UDA and 7) FedProx-UDA: naive combinations of FedAvg/FedProx with UDA. 8) FedAvg-FixMatch and 9) FedProx-FixMatch: naive combination of with FixMatch with FedAvg/FedProx. For training, we use SGD with adaptive-learning rate decay introduced in (Serra et al., 2018) with the initial learning rate $1e-3$. We use ResNet-9 networks as the backbone architecture for all baselines and our methods. We ensure that all hyper-parameters are set equally for all base models and ours to perform fair evaluation and comparison. Please see the Section A in the Appendix for further details. For all experiments, we report the mean and the standard deviation over 3 runs.

6.2 Experimental Results

Results on Batch-IID & NonIID Tasks Table 1 shows performance comparison of our models and naive Fed-SSL algorithms on Batch-IID and NonIID tasks under the two different scenarios. We observe that our model outperforms all naive Fed-SSL baselines for all tasks and scenarios. In particular, under labels-at-server scenario, which is more challenging than labels-at-client scenario, we observe that the naive combination models significantly suffer from the forgetting issue and their performances keeps deteriorating after a certain communication round. This phenomenon is mainly caused by the base models failing to properly perform disjoint learning, in which case the learned knowledge from the labeled and unlabeled data causes inter-task interference. Contrarily, our methods show consistent and robust performance regardless where the labeled data exists, which shows that our decomposition techniques effectively handles the challenging disjoint learning scenario. In addition, when the class-wise distribution is imbalanced for each client (Non-IID task), we observe that the base models’ performance slightly drops by 1-3%, while our methods show consistent.
We conjecture that this is because, for streaming situation, the model may not sufficiently train on the labeled set at each training steps. As shown in Figure 6 (c), our method effectively preserves knowledge interference between supervised learning and unsupervised learning. Interestingly, our model without inter-client consistency loss still outperforms base models. This shows that our inter-client consistency loss is effective since there exists knowledge interference between supervised learning and unsupervised learning. For further analysis, we show the effect of each decomposed variables, $\sigma$ and $\psi$, in Figure 6 (b). Removing either of $\sigma$ and $\psi$ results in substantial drop in the performance, with larger performance degeneration when dropping $\sigma$, which captures much more essential knowledge from labeled data (Green line). Such decomposition is effective since there exists knowledge interference between supervised learning and unsupervised learning. We show this with an experiment where we perform semi-supervised learning with 5 labeled instances per class and 1,000 unlabeled instances for 100 rounds. We measure accuracy on the labeled set at each training steps. As shown in Figure 6 (c), our method effectively preserves disjoint learning, also effectively enhances model performance.

Table 2: Averaged Local Performance on Streaming-NonIID Task. We use 10 clients 100 rounds. We measure averaged local model accuracy and communication costs. Note that the SL models learn on both $S$ and $H$ with full labels, and are utilized as the upper bounds for each experiment.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Labels-at-Client Scenario</th>
<th>Labels-at-Server Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc. (%)</td>
<td>S2C Cost</td>
</tr>
<tr>
<td>Local-SL</td>
<td>87.29 ± 0.36</td>
<td>N/A</td>
</tr>
<tr>
<td>Local-UDA</td>
<td>76.07 ± 0.28</td>
<td>N/A</td>
</tr>
<tr>
<td>Local-FixMatch</td>
<td>62.46 ± 0.32</td>
<td>N/A</td>
</tr>
<tr>
<td>FedProx-SL</td>
<td>87.06 ± 0.26</td>
<td>100 %</td>
</tr>
<tr>
<td>FedProx-UDA</td>
<td>73.71 ± 0.17</td>
<td>100 %</td>
</tr>
<tr>
<td>FedProx-FixMatch</td>
<td>62.40 ± 0.43</td>
<td>100 %</td>
</tr>
<tr>
<td>FedMatch (Ours)</td>
<td>77.95 ± 0.44</td>
<td>97 %</td>
</tr>
</tbody>
</table>

Figure 6: Ablation Study and Additional Analysis on FedMatch Algorithm. We study effectiveness of each components of our method, (a) inter-client consistency loss and (b) parameter decomposition. (c) We effectively tackle the inter-task interference. (d) Performance improvement of our method when labeled data is increased.

This shows that our inter-client consistency loss effectively enhances consistency with the helper agents selected from server based on model similarity, which is good at, in particular, class-imbalanced tasks. We also visualize the test accuracy curve for our models and naive Fed-SSL in Fig. 5. Our method (Red line) trains faster and consistently outperforms the base models, and is most robustness against inter-task interference in both scenarios. For analysis on the averaged communication costs, please see Section B.1 in the Appendix.

Results on Streaming-NonIID Task. Table 2 shows averaged local model performance on Streaming-NonIID tasks with 10 synchronized clients. For the labels-at-client scenario, our proposed method outperforms local-SSL and naive Fed-SSL models with large margins, 4-15%\%, except for the SL models. There is no huge difference of performance between local SSL and Fed-SSL models, and this implies that our method effectively utilizes inter-client knowledge in this streaming setting. In the labels-at-server scenario, interestingly, the performance of FedProx-SL decreases by around 5%\% compared to the labels-at-client scenario, while Fed-SSL models obtain improved performance. We conjecture that this is because, for streaming situation, the model may not sufficiently train on the new data, while Fed-SSL models overcomes it by utilizing only the consistent pseudo-labels. Even on this task, FedMatch outperforms all baselines with significantly smaller communication cost on average (see Section B.1 for detailed analysis of the averaged communication costs).

Effectiveness of Inter-Client Consistency. To show the effectiveness of our inter-client consistency loss, we eliminate the loss, while learning on Batch-IID task with 100 clients ($F=0.05$). In Figure 6 (a), when we remove our inter-client consistency loss, we observe that the performance has slightly dropped (Pink line) from one with the loss term (Red line). This gap clearly tells us that our inter-client consistency loss improves model consistency across multiple models while keeping reliable knowledge. Interestingly, our model without inter-client consistency loss still outperforms base models. This additionally implies that our another proposed method, parameter decomposition for disjoint learning, also effectively enhances model performance.

Effectiveness of Parameter Decomposition. Our model with parameter decomposition alone, without inter-client consistency loss, outperforms base models. For further analysis, we show the effect of each decomposed variables, $\sigma$ and $\psi$, in Figure 6 (b). Removing either of $\sigma$ and $\psi$ results in substantial drop in the performance, with larger performance degeneration when dropping $\sigma$, which captures much more essential knowledge from labeled data (Green line). Such decomposition is effective since there exists knowledge interference between supervised learning and unsupervised learning. We show this with an experiment where we perform semi-supervised learning with 5 labeled instances per class and 1,000 unlabeled instances for 100 rounds. We measure accuracy on the labeled set at each training steps. As shown in Figure 6 (c), our method effectively preserves disjoint learning, also effectively enhances model performance.
learned knowledge from labeled set, while other base models suffer from knowledge interference. This effective separation of supervised and unsupervised learning tasks enhances the overall performance of our methods even without inter-client consistency loss as shown in Figure 6 (a) (shown in pink).

**Number of Labels per Class** We increase the number of labels per class for each client in a range of 1, 5, 10, and 20 on Batch-IID task (CIFAR-10). Our method shows consistent performance improvement as the number of labels increases. Interestingly, we observe that baseline models, FedProx-UDA/FixMatch, show performance degradation even when the labeled data increases (5 → 10). These results show that our method effectively utilize knowledge from labeled and unlabeled data in federated semi-supervised learning settings, while other naive combinations of FSSL could fail to learn properly from labeled and unlabeled data in federated learning framework.

7 RELATED WORK

**Federated Learning** A variety of approaches for averaging local weights at server have been introduced in the past few years. FedAvg (McMahan et al., 2017) performs weighted-averaging on local weights according to the local train size. FedProx (Li et al., 2018) uniformly averages the local updates while clients perform proximal regularization against the global weights, while FedMA (Wang et al., 2020) matches the hidden elements with similar feature extraction signatures in layer-wise manner when averaging local weights. PFNM (Yurochkin et al., 2019) introduces aggregation policy which leverages Bayesian non-parametric methods. Beyond focusing on averaging local knowledge, there are various efforts to extend FL to the other areas, such as continual learning under federated learning frameworks (Yoon et al., 2020a) inspired by parameter decomposition techniques proposed by (Yoon et al., 2020b). Recently, interests of tackling scarcity of labeled data in FL are emerging and discussed in (Jin et al., 2020; Guha et al., 2019; Albaseer et al., 2020).

**Semi-Supervised Learning** While there exist numerous work on SSL, we mainly discuss consistency regularization approaches. Consistency regularization (Sajjadi et al., 2016) assumes that the class semantics will not be affected by transformations of the input instances, and enforces the model output to be the same across different input perturbations. Some extensions to this technique perturb inputs adversarially (Miyato et al., 2018), through dropout (Srivastava et al., 2014), or through data augmentation (French et al., 2018). UDA (Xie et al., 2019) and ReMixMatch (Berthelot et al., 2019a) use two sets of augmentations, weak and strong, and enforce consistency between the weakly and strongly augmented examples. Recently, in addition to enforcing consistency between weak-strong augmented pairs, FixMatch (Sohn et al., 2020) performs pseudo-label refinement on model predictions via thresholding. Entropy minimization (Grandvalet & Bengio, 2004) which enforces the classifier to predict low-entropy on unlabeled data, is another popular technique for SSL. Pseudo-Label (Lee, 2013) constructs one-hot labels from highly confident predictions on unlabeled data and uses these as training targets inn a standard cross-entropy loss. MixMatch (Berthelot et al., 2019c) performs sharpening on target distribution on unlabeled data, to further refine the generated pseudo-label.

8 CONCLUSION

In this work, we introduced two practical scenarios of Federated Semi-Supervised Learning (FSSL) where each client learns with only partly labeled data (Labels-at-Client scenario), or supervised labels are only available at the server, while clients work with completely unlabeled data (Labels-at-Server scenario). To tackle the problem, we propose a novel method, Federated Matching (FedMatch), which introduces the inter-client consistency loss that aims to maximize the agreement between the models trained at different clients, and the parameter decomposition for disjoint learning which decomposes the parameters into one for labeled data and the other for unlabeled data for preservation of reliable knowledge, reduction of communication costs, and disjoint learning. Through extensive experimental validation, we show that FedMatch significantly outperforms both local semi-supervised learning methods and naive combinations of federated learning algorithms with semi-supervised learning on diverse and realistic scenarios. As future work, we plan to further improve our model to tackle the scenario where pretrained models deployed at each client adapts to a completely unlabeled data stream (e.g. on-device learning of smart speakers).
Acknowledgements  This work was supported by Samsung Research Funding Center of Samsung Electronics (No. SRFC-IT1502-51), Samsung Advanced Institute of Technology, Samsung Electronics Co., Ltd., Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2016M3C4A7952634), the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (2018R1A5A1059921), and Center for Applied Research in Artificial Intelligence (CARAI) grant funded by DAPA and ADD (UDI190031RD). Also, this work was supported by Institute of Information communications Technology Planning Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST))

REFERENCES
Abdullatif Albaseer, Bekir Ciftler, Mohamed Abdallah, and Ala Al-Fuqaha. Exploiting unlabeled data in smart cities using federated learning. 01 2020.


We use Stochastic Gradient Descent (SGD) to optimize our model with initial learning rate 1e-3. We also reimplement FixMatch algorithms with strong augmentation as RandAugment (Cubuk et al., 2019). We consider UDA (Xie et al., 2019) and FixMatch (Sohn et al., 2020) as our baselines, since they are state-of-the-art SSL models and are based on the consistency-based mechanisms that are conceptually similar to our inter-client consistency loss. We reimplement UDA with the Training Signal Annealing (TSA) and exponential scheduling for its best performance as reported in their paper (we use RandAugment (Cubuk et al., 2019) for consistency regularization with random magnitude). We also reimplement FixMatch algorithms with strong augmentation as RandAugment (Cubuk et al., 2019). For weak augmentation (flip-and-shift), however, as the performance has significantly dropped when we apply the weak augmentation, we use original images rather than weakly augmenting the images. We fix confidence threshold \( \tau = 0.85 \) for all FixMatch and our model experiments. For federated learning frameworks, we use FedAvg (McMahan et al., 2017) and FedProx (Li et al., 2018) algorithms since they are the standard baselines for federated learning and can be easily combined with the SSL baselines. Detailed hyper-parameter settings are described in Table 4.

A EXPERIMENTAL DETAILS

We describe our experimental setups in detail, such as our baseline models, network architecture that is used for all base models and our method, and the detailed training setups.

A.1 BASELINE MODELS

We consider UDA (Xie et al., 2019) and FixMatch (Sohn et al., 2020) as our baselines, since they are state-of-the-art SSL models and are based on the consistency-based mechanisms that are conceptually similar to our inter-client consistency loss. We reimplement UDA with the Training Signal Annealing (TSA) and exponential scheduling for its best performance as reported in their paper (we use RandAugment (Cubuk et al., 2019) for consistency regularization with random magnitude). We also reimplement FixMatch algorithms with strong augmentation as RandAugment (Cubuk et al., 2019). For weak augmentation (flip-and-shift), however, as the performance has significantly dropped when we apply the weak augmentation, we use original images rather than weakly augmenting the images. We fix confidence threshold \( \tau = 0.85 \) for all FixMatch and our model experiments. For federated learning frameworks, we use FedAvg (McMahan et al., 2017) and FedProx (Li et al., 2018) algorithms since they are the standard baselines for federated learning and can be easily combined with the SSL baselines. Detailed hyper-parameter settings are described in Table 4.

A.2 NETWORK ARCHITECTURE

We build ResNet-9 networks as our base architecture for all base model and our method. In the architecture, the first two convolutional neural layers have 64 and 128 filters and the same 3 x 3 kernel sizes followed by 2 x 2 max-pooling layer. Then we have a skip connection between the subsequent two convolution layers with 128 filters. We then double the filter size from 128 to 256 with the next conv layer and down-sample via the following 2 x 2 max-pooling layer. We repeat the previous step, such that we have 512 filter size and 4 x 4 kernel size. Then, we perform another skip connection through the two subsequent conv layers with 512 filters. As a final step, we down-sample the kernel size from 4 x 4 to 1 x 1, then perform softmax classifier with the last fully connected layer. All layers are equally initialized based on the variance scaling method. The model architecture is described in Table 3.

A.3 TRAINING DETAILS

We use Stochastic Gradient Descent (SGD) to optimize our model with initial learning rate 1e-3. We also adopt adaptive learning rate decay which is introduced by (Serra et al., 2018). The learning rate strategy gradually reduces the learning rate by a factor of 3 for every 5 epochs that validation loss does not consecutively decreases. We use L2 weight decay regularization on the base architecture with L2 factor to be 1e-4. All hyper-parameters and other training setups are equally set for fair comparison as shown in Table 4. In the table, we denote LPC as number of labels per class for each

<table>
<thead>
<tr>
<th>Layer</th>
<th>Filter Shape</th>
<th>Stride</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>N/A</td>
<td>N/A</td>
<td>32 x 32 x 3</td>
</tr>
<tr>
<td>Conv 1</td>
<td>3 x 3 x 3 x 64</td>
<td>1 x 1</td>
<td>32 x 32 x 64</td>
</tr>
<tr>
<td>Conv 2</td>
<td>3 x 3 x 64 x 128</td>
<td>1 x 1</td>
<td>32 x 32 x 128</td>
</tr>
<tr>
<td>Pool 1</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>16 x 16 x 128</td>
</tr>
<tr>
<td>Conv 3</td>
<td>3 x 3 x 128 x 128</td>
<td>1 x 1</td>
<td>16 x 16 x 128</td>
</tr>
<tr>
<td>Conv 4</td>
<td>3 x 3 x 128 x 256</td>
<td>1 x 1</td>
<td>16 x 16 x 256</td>
</tr>
<tr>
<td>Pool 2</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>8 x 8 x 256</td>
</tr>
<tr>
<td>Conv 5</td>
<td>3 x 3 x 256 x 512</td>
<td>1 x 1</td>
<td>8 x 8 x 512</td>
</tr>
<tr>
<td>Pool 3</td>
<td>2 x 2</td>
<td>2 x 2</td>
<td>4 x 4 x 512</td>
</tr>
<tr>
<td>Conv 6</td>
<td>3 x 3 x 512 x 512</td>
<td>1 x 1</td>
<td>4 x 4 x 512</td>
</tr>
<tr>
<td>Pool 4</td>
<td>4 x 4</td>
<td>4 x 4</td>
<td>1 x 1 x 512</td>
</tr>
<tr>
<td>Softmax</td>
<td>N/A</td>
<td>N/A</td>
<td>1 x 1 x 10</td>
</tr>
</tbody>
</table>

Table 3: Network Architecture of ResNet-9
We split the dataset \( D \) into a set of labeled data \( S \) and a set of unlabeled data \( U \). \( U \) is divided into \( K \) subsets which are distributed to \( K \) clients (Batch Task). For streaming tasks, we further split all instances in each subset into \( T \) subsets for \( T \) streaming steps. For class-imbalanced tasks, we additionally control the number of instances per class for each client.

Table 4: Hyper-Parameters & Training Setups

We provide all hyper-parameters and training setups for all baseline models and our method. Detailed hyper-parameters are also available in the code.

<table>
<thead>
<tr>
<th>Methods</th>
<th>lr</th>
<th>wd</th>
<th>( \lambda_s )</th>
<th>( \lambda_u )</th>
<th>( \lambda_{ICS} )</th>
<th>( \lambda_{l1} )</th>
<th>( \lambda_{l2} )</th>
<th>LPC</th>
<th>( E^S_{client} )</th>
<th>( E^U_{client} )</th>
<th>( E^S_{server} )</th>
<th>( \mu )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL</td>
<td>1e-3</td>
<td>1e-4</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>-</td>
<td>1e-2</td>
</tr>
<tr>
<td>UDA</td>
<td>1e-3</td>
<td>1e-4</td>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>-</td>
<td>1e-2</td>
</tr>
<tr>
<td>FixMatch</td>
<td>1e-3</td>
<td>1e-4</td>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>-</td>
<td>1e-2</td>
</tr>
<tr>
<td>FedMatch</td>
<td>1e-3</td>
<td>1e-4</td>
<td>10</td>
<td>1e-2</td>
<td>1e-4</td>
<td>1e-5</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Labels-at-Server Scenario**

<table>
<thead>
<tr>
<th>Methods</th>
<th>lr</th>
<th>wd</th>
<th>( \lambda_s )</th>
<th>( \lambda_u )</th>
<th>( \lambda_{ICS} )</th>
<th>( \lambda_{l1} )</th>
<th>( \lambda_{l2} )</th>
<th>LPC</th>
<th>( E^S_{client} )</th>
<th>( E^U_{client} )</th>
<th>( E^S_{server} )</th>
<th>( \mu )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL</td>
<td>1e-3</td>
<td>1e-4</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>100</td>
<td>100</td>
<td>1e-2</td>
</tr>
<tr>
<td>UDA</td>
<td>1e-3</td>
<td>1e-4</td>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>100</td>
<td>100</td>
<td>1e-2</td>
</tr>
<tr>
<td>FixMatch</td>
<td>1e-3</td>
<td>1e-4</td>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>100</td>
<td>100</td>
<td>1e-2</td>
</tr>
<tr>
<td>FedMatch</td>
<td>1e-3</td>
<td>1e-4</td>
<td>10</td>
<td>1e-2</td>
<td>1e-5</td>
<td>10</td>
<td>100</td>
<td>-</td>
<td>100</td>
<td>100</td>
<td>-</td>
<td>1e-2</td>
</tr>
</tbody>
</table>

client (or at server). \( B^S \) and \( B^U \) denote batch-size of labeled set \( S \) and unlabeled set \( U \). \( \mu \) is a hyper-parameter for FedProx framework. We additionally provide visual illustration of our dataset configuration. Please see Figure 7.

B ADDITIONAL ANALYSIS AND EXPERIMENTAL Results

In this section, we additionally provide more analysis and experimental results, such as analysis on communication costs and number of labels per class, experiments on real-world dataset, different backbone architecture, fraction of clients per communication round.

B.1 THE EFFICIENT COMMUNICATION OF FedMatch

Since the actual bit-level compression techniques are rather implementation issues, which are beyond our research scope, we only consider the reduction of the amount of information that needs to be transmitted between the server and the client. To minimize the communication costs, we not only learn \( \psi \) to be sparse, but also subtract the parameters between server and client, such that \( \Delta\psi = \psi^l - \psi^G \) and \( \Delta\sigma = \sigma^l - \sigma^G \), then send only the difference, \( \Delta\psi \) and \( \Delta\sigma \), as sparse matrices from both directions of server-to-client (S2C) and client-to-server (C2S). Here, S2C and C2S costs are the sums of \( \Delta\sigma \) and \( \Delta\psi \). When transmitting the difference for each parameter to either way, we discard almost unchanged values in an element-wise manner, so that only meaningful neural values can be updated either server- or client-side. We observe that the range of the threshold values is from 1e-5 to 5e-5, such that the model performance is well-preserved and not significantly harmed, while maximizing the reduction of communication costs.

As shown in Figure 8, we observe that both the S2C and C2S costs are gradually decreased during the learning phases on both batch and streaming datasets under labels-at-client (Figure 8 (a) and (b)) and labels-at-server scenarios (Figure 8 (c) and (d)). This is because each parameter separately learns different tasks (i.e. supervised and unsupervised learning) effectively, which results in rapid convergence to optimal points, respectively. Further, for the labels-at-server scenario, since labeled data is not available at client, client even does not need to transfer \( \Delta\sigma \) to the server (see Figure 8 (c) right and (d) right), which is extremely efficient than the labels-at-client scenario where both \( \Delta\sigma \) and \( \Delta\psi \) must be transferred to the server. For both scenarios, indeed, S2C contains the cost...
of helper agents, such that \( \Delta \psi_1^{1:H} = \sum_{j=1}^{H} \psi_j^1 - \psi_j^0 \). However, as shown in Figure 8, transmitting multiple helper agents (\( H=2 \) in our experiments) does not significantly affect the total S2C costs thanks to our novel decomposition techniques as well as efficient subtracting method, such that model reconstruction can be possible without meaningful information loss at either server- or client-side.

**B.2 FURTHER ANALYSIS OF THE NUMBER OF LABELS PER CLASS**

We explain our analysis of the number of labels per class under Section 6.2, and here we further provide additional experimental results. We conduct experiments with our method without the decomposition technique. As shown in Table 5, our method without decomposition technique (indicated as FedMatch (w/o)) shows not much performance improvement when the number of labels per class increases from 5 to 10 (around 3.3%\%p) than from 1 to 5 (around 9.3%\%p) and from 10 to 20 (10.0%\%p), which are the similar tendency with the baseline models in Table 5. However, with the decomposition technique, our method shows consistent performance improvement, which implies that our proposed technique has the effectiveness to handle inter-task interference and preserve reliable knowledge in the novel federated semi-supervised learning scenarios.
1341 images from patients diagnosed of viral pneumonia. We use 10 clients with a fraction of 1.0 with the similar tendency with that of the results based on ResNet-9. This shows that our methods AlexNet-Like (Serra et al., 2018), of which the first three layers are convolutional neural layers with (communication rate). We use 5 labeled examples per class for each client, leaving the rest of the

to see the effect of participation rate of clients, we increase the fraction of available clients per com-
to show our method consistently work with real-world dataset, we further conduct experiment on
Performance Comaprison utilizing AlexNet-Like architecture Table 6: Experiments based on AlexNet-Like Architecture

<table>
<thead>
<tr>
<th>Methods</th>
<th>Streaming-NonIID (F=1.0)</th>
<th>Batch-IID (Labels-at-Client)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Labels-at-Client</td>
<td>Labels-at-Server</td>
</tr>
<tr>
<td>Acc. (%)</td>
<td>Acc. (%)</td>
<td>F=0.05</td>
</tr>
<tr>
<td>FedAvg-SL</td>
<td>68.20 ± 0.29</td>
<td>70.51 ± 0.11</td>
</tr>
<tr>
<td>FedProx-SL</td>
<td>68.47 ± 0.13</td>
<td>70.55 ± 0.72</td>
</tr>
<tr>
<td>FedAvg-UDA</td>
<td>52.25 ± 0.04</td>
<td>46.28 ± 0.32</td>
</tr>
<tr>
<td>FedProx-UDA</td>
<td>52.84 ± 0.15</td>
<td>46.35 ± 0.31</td>
</tr>
<tr>
<td>FedAvg-FixMatch</td>
<td>57.09 ± 0.89</td>
<td>52.67 ± 0.78</td>
</tr>
<tr>
<td>FedProx-FixMatch</td>
<td>57.12 ± 0.41</td>
<td>51.51 ± 0.32</td>
</tr>
<tr>
<td>FedMatch (Ours)</td>
<td>65.56 ± 0.18</td>
<td>59.12 ± 0.35</td>
</tr>
</tbody>
</table>

B.3 Experiments on Real-world Dataset
To show our method consistently work with real-world dataset, we further conduct experiment on COVID-19 Radiography Dataset (Chowdhury et al., 2020) which is a real-world dataset that consists of X-ray images from COVID and non-COVID patients. The COVID-19 dataset contains 219 X-ray images from the patients diagnosed of COVID-19, 1341 images from normal (healthy) patients, and 1341 images from patients diagnosed of viral pneumonia. We use 10 clients with a fraction of 1.0 (communication rate). We use 5 labeled examples per class for each client, leaving the rest of the image as unlabeled. We find this setting to be realistic as the datasets are, since we may not have skilled radiologists that can fully label the X-ray images taken at the local hospitals. We compared our method against baselines which naively combine semi-supervised learning and federated learning models during training 100 rounds. As shown in the left table of Figure 9, our method consistently outperforms all base models with large margins (around 4%~10%) in both scenarios. The test accuracy curves in Figure 9, we can see that our method trains faster than the base models and shows more stability during training. We believe that these additional experimental results further strengthen our paper.

B.4 Backbone Architecture
Most existing works on federated learning considers smaller networks since the focus is on-device learning of low-resource devices, and thus we utilize a smaller backbone networks than ResNet-9. To verify that our method also successfully works on the smaller & different architecture, we adopt AlexNet-Like (Serra et al., 2018), of which the first three layers are convolutional neural layers with 64, 128, and 256 filters with the 4, 3, and 2 kernel sizes followed by the two fully-connected layers of 2048 units, while 2 × 2 max-pooling layers are followed after each convolutional layer. In Table 6, for both Streaming-NonIID and Batch-IID tasks, our methods still outperforms all naive Fed-SSL models with the similar tendency with that of the results based on ResNet-9. This shows that our methods can be applied to the smaller and different base networks, and still effectively utilize inter-client and reliable knowledge across multiple clients than naive algorithms.

B.5 Fraction of Available Clients per Communication Round
To see the effect of participation rate of clients, we increase the fraction of available clients per communication round in a range of 0.05, 0.10, and 0.2. This means, for every round, server can connect to the arbitrary 5, 10, and 20 available clients out of 100 clients, and perform distributed learning on each individual local data through each client and updates global knowledge by aggregating the locally-learned knowledge. The experimental results are shown in Table 6 Batch-IID. We observe that the performances of all models are slightly improved when the fraction increases. This is natural that the more knowledge the client updates, the more the global performance is improved. We are not able to find any extraordinary phenomenon on the fraction of the number of clients per round.