Using Transfer Learning in Building Federated Learning Models on Edge Devices

Jordan Suzuki
Mount Royal University
Calgary, Canada
jsuzu530@mtroyal.ca

Saba F. Lameh
University of Tabriz
Tabriz, Iran
farshbaf.saba98@ms.tabrizu.ac.ir

Yasaman Amannejad
Mount Royal University
Calgary, Canada
yamannejad@mtroyal.ca

***Abstract*—Today's state-of-the-art machine learning techniques, such as deep networks, are typically trained using cloud platforms, leveraging the elastic scalability of the cloud. For such processing, data from various sources need to be transferred to a cloud server. While this works well for some application domains, it is not suitable for all due to privacy and network overhead concerns. Federated learning is a machine learning setting designed for learning from distributed data over edge devices. In federated learning, data does not leave users' devices. All users collaboratively train a model without sharing their data. Each user trains a local model with their data and shares the model updates with a federated server to aggregate and build a global model. Such training respects users' privacy while reducing the network overhead for transferring data to a central server. Despite all advantages, each user may not have limited data which might not be enough for training their local model. To solve this challenge while keeping the client data still private, we have combined transfer learning with federated learning where we train a base model with a different public dataset. The base models then are passed to federated user to be fine tuned for the final target tasks. Using transfer learning we have built federated image classification model for CIFAR-100 dataset and have studied how different parameters affect the model accuracy.**

***Keywords*— Distributed Machine Learning, Federated Learning, Transfer Learning, Edge Computing**

I. Introduction

Learning from data dispersed over billions of smart user devices has amassed a huge interest in recent years. Smart user devices and sensors are continuously collecting massive amounts of data. The data often contains sensitive or private information, e.g., smart home cameras may contain life-logging videos. In order to preserve the privacy of users or organizations to follow data privacy protection rules such as GDPR [1] the data needs to be kept private and decenteralized. This prevents data scientists and machine learning engineers from training models using traditional machine learning techniques due to the lack of access to this private data. The need to process distributed private data as well as the increasing availability of processing power in smart devices has led to a growing interest in distributed machine learning models that can be trained on distributed edge devices.

Federated learning [2] is a machine learning setting where many clients, e.g. user devices, collaboratively train a model under the coordination of a central server, e.g. federated server or the service provider, while keeping the training data decentralized. The raw data generated or collected at users' devices serve as the training data. Users' devices work cooperatively to train a complex machine learning model without sharing their private data. Instead, each device trains a copy of the same machine learning model locally using its own private data and only shares the intermediate data, i.e., parameters of the trained local model with a server. The federated server aggregates local model parameters into a global model and sends back the aggregated model to all mobile devices for further training. This process continues iteratively in multiple rounds until a desirable accuracy for the learning objective is achieved or a training budget is hit.

Federated learning mitigates many of the systemic privacy risks and costs associated with traditional and centralized machine learning approaches: 1) Keeping data local on users' devices addresses the fundamental concern of privacy and data security. 2) Moreover, it allows exploiting the growing processing power of edge devices rather than relying on only powerful machine learning servers. 3) Transferring all the data from users' devices to a remote cloud server is not always feasible due to the limiting factors such as network bandwidth of these devices and the long propagation delays that can incur unacceptable latency. Federated learning avoids such latency by conducting machine learning on clients' devices. 4) With the data staying local on user devices, federated learning avoids burdening the backbone networks to transfer all data to a central server.

Despite all the advantages, training a model in federated fashion may raise new challenges. Federated clients collectively have access to a large training set, however, each client only has access to a subset of the dataset. In each round of the federated training, users train a copy of the same model. When training a machine learning model for a complex task, e.g., an image classification model for a dataset with many classes of data, the small dataset owned by each client may not be enough for training a well-performing local model. In such cases, clients need to do more rounds of parameter exchange with the federated server to collectively converge to a well-performing model. This can result in a lengthy training process. In many real-word scenarios where obtaining training data is difficult or expensive, practitioners have applied transfer learning. Transfer learning is a machine learning method where a model, i.e., a base model, is trained once on a large and generic dataset and its weights are then used as the starting point for a new model, i.e., target model, on a different task. Transfer learning is applied successfully in many domains. We believe that this technique can be combined with federated learning to address the challenge of limited client data for training complex models.

In this paper, we aim to train a federated machine learning model for a complex image classification task. The complexity of the task at hand arises from the high number of image classes, i.e., 100 fine and 20 coarse labels, and low resolution color images. A deep learning network that can perform well for such complex task may require millions of parameters. Learning a large number of parameters using a relatively small data size at each client may require clients to continue the federated training process for many rounds. To avoid lengthy training rounds, we aim to use transfer learning in our federated training process. Through this study, we address the following two research questions:

- **RQ1** - Can we use transfer learning in the federated learning process to reduce the training time? and what is the effect of the base model used in transfer learning?
- **RQ2** - What are the effects of the federated parameters used in the accuracy of the federated model?

To answer these question we train a federated model for classifying CIFAR-100 dataset [3] using 4 base models with 2 different architectures trained on 2 different source datasets. We vary the parameters of the federated process and analyze the effect of the base model and the federated parameters on the accuracy of the trained federated models. Our results shows the effectiveness of transfer learning for training a complex federated model. A larger base model is effective only when the source data is large enough to train the model. The absence of some clients in training rounds due to the power outage of the device does not effect the model accuracy negatively, when there is enough clients to replace the previously participating clients. A dataset with larger data labels can increase the model complexity. Using a larger model or more number of clients can help the model to compensate the decrease in the model accuracy.

The remainder of the paper is organized as follows. In Section II, we explain federated learning and transfer learning concepts. Related work are discussed in Section III. We describe our methodology and the experiment setup in Section IV. The results of our analysis are discussed in Section V.

II. Background

A. Federated Learning

The term federated learning was introduced in 2016 by McMahan et al. [2]: "We term our approach *Federated Learning*, since the learning task is solved by a loose federation of participating devices (we refer to as clients) which are coordinated by a central server." The goal of federated learning is to provide a framework for distributed ML on unbalanced and non identically and independently distributed (non-IID), data partitioned over client devices with limited communication bandwidth.

In general, there are two main entities in federated learning process: the data owners, i.e., *client*, and the global model owner, i.e., *server*. Let $Clients = \{client_1, client_2, ..., client_n\}$ denote the set of n clients, each of which has a dataset D_i, $i \in n$, stored on their personal devices, e.g., cellphones. In classical approaches, all clients send their data to the server and the server trains a conventional model, $model$, in central fashion using all data $D = \cup_{i=1}^{n} D_i$. However this is not possible for all applications due to privacy concerns and network bandwidth limitations of the edge computing devices. In federated learning, clients do not share their private data with the remote server, instead, they build a local model, $model_i$, using their own data, D_i, and share the parameters of their trained model with the federated learning server. The server collects all local model parameters and aggregates them to build a global model, $model_{federated}$. The trained model is sent back to clients for further enhancements. This process continues for r rounds of training.

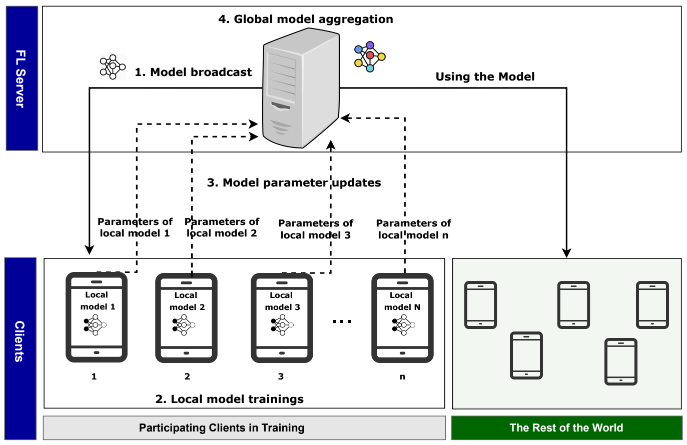

FL Server
4. Global model aggregation
1. Model broadcast
Using the Model
3. Model parameter updates
Parameters of local model 1
Parameters of local model 2
Parameters of local model 3
Parameters of local model n
Clients
Local model 1
Local model 2
Local model 3
Local model N
1
2
3
n
2. Local model trainings
Participating Clients in Training
The Rest of the World

Fig. 1: The main steps in federated process

A typical training process is shown in Fig. 1 and includes the following four steps:

1) **Model broadcast**: The server defines the structure of the model to be trained by all clients and decides the hyper-parameters of the local and global models, e.g., the optimizer or the learning rate of a neural network. Next, the server broadcasts the initial model or the model aggregated from the local training of clients.
2) **Local model training**: Clients participating in the training process download the model and train the model based on their local data. The objective of each client during its local training is to find optimal model parameters that minimize the model loss based on client's local data.
3) **Model parameter updates**: When each client finishes its training, they share their trained model with the federated server.
4) **Global model aggregation**: When the participating clients share their models with the server, the server aggregates them into a global model. The server's objective is to minimize the global loss function.

It should be mentioned that during the training process, the number of participating clients may vary. Some clients might not be available to participate in the training process, e.g., their device might be turned off. Therefore, in each round of the training process, the server only expects to get model updates from a subset of all clients, $C \subset Clients$. When the model is trained, it can be used by the rest of the clients, even those who did not participate or partially participated in the training process.

B. Transfer Learning

Transfer learning is a technique used in machine learning to improve a learner model in one domain by transferring information from a related domain. Formally, given a source domain D_S with a corresponding source task T_S and a target domain D_T with a corresponding task T_T, transfer learning is the process of improving the target predictive function $f_T(\cdot)$ by using the related information from D_S and T_S, where $D_S \neq$ D$_T$ or $T_S \neq$ T$_T$ [4]. This technique is specially useful for training model for domains where training data is difficult or expensive to obtain. For example, for an image classification task, where limited D_T is available, a deep learning model can be trained on D_S to conduct image classification for D_S. When building an image classification model for D_T, this initial model is used as the *base model*. The target image classification model for D_T is then constructed by appending additional final layer(s) to the network. This way the only parameters to train for the target model will be the parameters of the final layer(s) which are normally less than the parameters of the whole model and can be learned by using limited training data available for the target domain. The reason transfer learning works is that in machine learning tasks as in image classification, there are some common patterns to learn, e.g., line, shape and edge detection in images, and the knowledge learned from other domains can be applied to the target domain with high accuracy and the limited data in the target domain can be used to refine the learned knowledge and also learn the domain specific patterns and parameters.

We believe that a baseline learned from related domains can provide a wise starting point for federated clients to use their own limited data to only learn the domain specific parameters and hence speed up the federated learning process. Therefore, in this work we aim to combine federated learning with transfer learning and observe the effect of the base model and federated parameter selection on the accuracy of the trained model.

III. Related work

Privacy-preserving data analysis has been studied for more than 50 years, but the federated learning setting has only been introduced in the last decade [2]. Federated learning was initially introduced with an emphasis on mobile and edge device applications [337, 334]. Nowadays, it is used for learning on users' mobile and edge devices as well as learning on multiple organizations' servers. Google and Apple are using federated learning in their Gboard and QuickType keyboards [376, 222, 491, 112, 383, 25] and many successful federated applications are developed in other domains such as vehicular networks [5, 6], mobile packet classification [7], cyberattack detection [8–10], finance risk prediction for reinsurance [476], pharmaceuticals discovery [179], electronic health records mining [184], medical data segmentation [15, 139], and smart manufacturing [354].

The growing demand for federated learning has resulted in a number of tools and frameworks becoming available [11–18]. NVIDIA Clara [11], PaddleFL [12] and FATE [13] are open-source frameworks that support federated learning. NVIDIA Clara is a healthcare application framework for imaging that makes it easy for developers to build, manage, and deploy intelligent medical imaging workflows and instruments. PaddleFL allows users to deploy a federated system in distributed clusters. FATE [13] is an open-source project which aims to provide a secure computing framework that will support the federated AI ecosystem. It implements multiple secure computation protocols in compliance with data protection regulations. To help users build their own federated models, programming libraries such as PySyft [14] and TensorFlow Federated [18] and simulation tools [15, 16, 18], and benchmark datasets are provided [16]. In this work, we use TensorFlow Federated [18] to implement our federated image classification model.

Initially, the focus of federated learning was on resource constrained edge devices which struggled with limited power and network bandwidth. This poses special challenges that require data scientists and machine learning engineers working in this area to have a special focus on the performance optimization of the developed models. Some researchers have focused on optimizing the performance of the local training and the parameter aggregations on the server [2, 19–22]. Nishio and Yonetani [23] have focused on the effect of the participating clients in the training process and have proposed a client selection strategy that filters out slow clients based on the estimation of the clients' execution time. Others have focused on optimizing the communication overhead in the federated process using learning from compressed models [2], adding a feedback loop regarding the global tendency of model updating [24], or filtering noisy data [25]. Inline with this direction of research, we aim to reduce the communication overhead in the federated process by reducing the number of federated rounds required when building a well-performing model. We accomplish this by using transfer learning and providing the clients with an initial base model trained on another domain.

As previously mentioned, the need for transfer learning occurs when there is a limited supply of target training data. This could be due to the data being rare, the data being expensive to collect and label, or the data being inaccessible. The existence of many public datasets makes transfer learning solutions an enticing approach. Transfer learning is successfully applied in many machine learning applications such as text classification [26–29], image classification [30–32], or in more specific application domains such as human activity classification [33], or software defect classification [34]. The base models we use in this work are based on the networks designed by Kolesnikov et al. [35].

They have designed two main components of scale and also group and standard normalization for training their base models and provided a recipe for hyper-parameter tuning of the models on the target task. They have shown the effectiveness of transferring their learning to multiple different tasks.

Previous work which has used transfer learning in the federated setting have used it for two parties with datasets that only have partial shared features. This can be useful for two organizations with different businesses where only a small portion of the feature space from both parties overlap. In this case, transfer learning techniques can be applied to provide solutions for the entire sample and feature space. Specially, a common representation between the two feature spaces is learned using common sample sets and later refined to obtain predictions for samples of each party with their full features. The goal of combining federated learning and transfer learning in our study is different than what we have used it for. We use transfer learning to train the base model that is passed to all clients as their starting point to reduce the number of the training rounds in the federated learning process. We explore the feasibility of combining these two learning methods as well as studying the effect of the base model architecture and training datasets.

IV. Methodology and Study Setup

A. Overview

Fig. 2 shows the components involved in our study. We train federated image classification deep learning networks for CIFAR-100 dataset. This dataset is explained in more detail in Section IV-C. To use this dataset in the federated setting, we use the distributed version of this dataset and assign each clients a subset of the whole dataset. To train the deep networks, we use transfer learning. We take a base model trained on a different dataset and then fine tune the model on our target CIFAR-100 dataset. We use four different base models and these models are described in Section IV-B. Since these models are trained on different datasets, we add new layers to the end of these models and use them as the initial models for local training. Each client re-trains a copy of the model to fine-tune the final layer using their own data. The models trained by all clients then get aggregated through the federated process described earlier in Section II-A.

B. Base Models

To train our image classification model, we transfer the learning of four BiT [2] models $S-r50\times 1$, $M-r50\times 1$, $S-r101\times 1$ and $M-r101\times 1$. The naming of the BiT models identifies three different characteristics of these models. The first part, S and M, shows the dataset that the base model is pre-trained on. S models are trained on ILSVRC-2012 variant of ImageNet, with 1.28 million images and $1,000$ classes. Each image has a single label. M models are pre-trained on the full ImageNet-21k dataset [36], which contains 14.2 million images and 21k classes organized by the WordNet hierarchy. Images may contain multiple

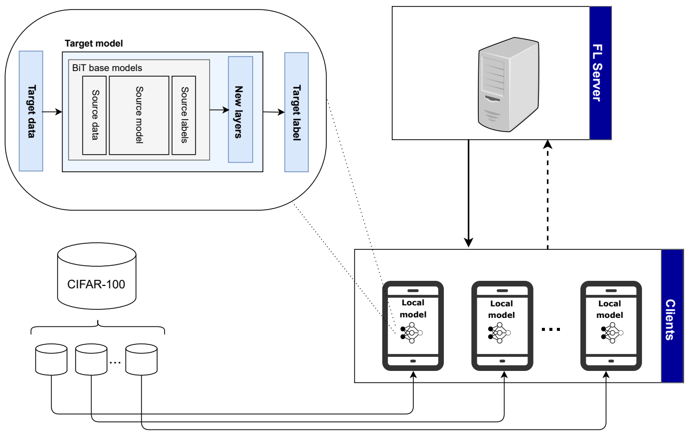


Fig. 2: Overview of our study setup

labels. Both datsets are used for computer vision tasks therefore suite the application of image classification for the CIFAR-100 set we use in this study. The second part which starts with r represents the layers contained in the Residual Network (ResNet) model being trained, and the number which comes after $\times$ shows the widening factor of the ResNet model.

C. CIFAR-100 Dataset

The dataset used in this work is derived from the CIFAR-100 dataset [3]. CIFAR-100 is a common benchmark dataset used for evaluating image classification models. This dataset consists of $60,000$ colour images of 32×32 with 100 classes 600 images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label, the class to which it belongs, and a "coarse" label, the superclass to which it belongs.

For the distributed federated model we use CIFAR-100 dataset available at [37]. In this dataset, all images are distributed among 500 training clients and 100 test clients. No clients share any data samples, so it is a true partition of CIFAR-100. The training clients form a true partition of the CIFAR-100 training split, while the test clients form a true partition of the CIFAR-100 testing split. The data partitioning is done using a two-stage hierarchical Latent Dirichlet Allocation process. The process has resulted in a data which is not independent and identically distributed, i.e., the data distribution is non-IID. This is consistent with the federated learning assumptions that different clients may collect datasets that have different distribution than datasets owned by other clients. In federated CIFAR-100 dataset, all clients have the same number of data points, but the distribution of the image labels among clients are very different. Figure 3 shows an example of the coarse label distribution among 4 clients in federated CIFAR-100 dataset.

D. Implementation

To implement our federated image classification network for CIFAR-100 dataset, we use the Google Colab Pro platform [38] which gives us priority access to use GPU

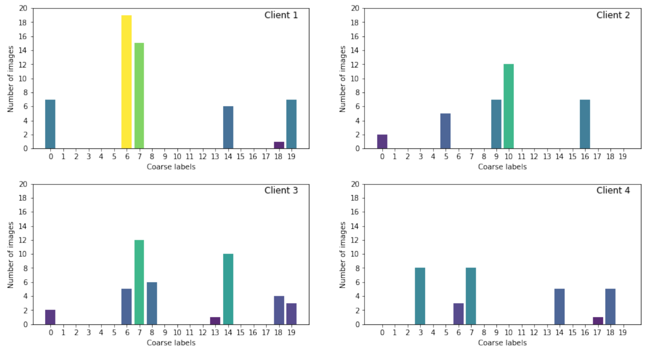


Fig. 3: Non-IID coarse label distribution among 4 clients

machines with about 24 GB of RAM. To implement the deep learning network we use Tensorflow 2.0 and the Keras library [39]. For implementing the federated model, we use the TFF library [18]. When training our deep learning networks we use the parameters shown in Table I. Some of these such as the number of Epochs are common hyper-parameters in training a deep learning network, some parameters such as Schedule length are specific for BiT models. These values are kept constant in all our experiments. The factors we vary among experiments are explained in Section IV-E. We have made all the code used in this work publicly available [40].

TABLE I: Constant parameters in our setup

Parameter	**Value**
Epochs	5
Steps per Epoch	10
Batch Size	128
Schedule length	10000 $\times$ 512 / Batchsize
Image resize	160 px
Image crop size	128 px
# of test clients	30

E. Experiment Factors

To conduct our study and answer the research questions outlined earlier in the paper, we vary a set of parameters in the experiments we run. Table II lists the experiment factors we vary and their different levels. The default levels used in the experiments are shown in bold format.

TABLE II: Experiment factors

Factor	**Value**
Base model	S-r50×1, **M-r50×1**, S-r101×1, m-r101×1
Training rounds	r = {25, 50, 75, 100, 125, 150, 175, 200}
# of training clients	c = {5, **10**}
Client selection	Constant, **Variable**
Label granularity	**Coarse**, Fine

We use four BiT models, $S-r50\times1$, $M-r50\times1$, $S-r101\times1$ and $M-r101\times1$, as the base models in training our federated image classification models. These models are relatively smaller models among the models introduced in [35] and hence more suitable for distributed learning on edge devices. When studying the effect of base model, we use all four models in our analysis. For studying the effect of the number of clients and the number of labels we use our default $M-r50\times1$ model.

In all experiments, each model is trained for 200 rounds and after every 25 rounds, we take a snapshot of the model and test it on our test clients. We report the results for all selected training rounds and hence none of the r values are set as our default value. We test the models on 30 test clients. The test clients are selected randomly from 100 test clients in CIFAR-100 dataset. For easier comparison among experiments, we use the same 30 test clients in all our experiments.

For clients participating in the training of the federated model we vary the number of clients as well as the client selection strategy. When training the models in federated setting, we use 5 and 10 clients as our training clients. The number of clients we us is relatively a small portion of 500 training clients in CIFAR-100 dataset. This is due to the memory limitation in Google Colab environment we use for our testing, but this is enough for us to answer our research questions. When running the models on real distributed user devices the number of clients can be increased to get higher accuracy. Each training client we use in our dataset has the same number of data points to train their local model. However, as we show earlier in Fig. 3, the distribution of the labels among clients is non-IID and each client may see a different subset of all labels available in the dataset. To select clients participating in the training rounds, we use two different client selection strategies. We investigate the accuracy of a federated model where clients participating in each round of training varies and we compare it with a model trained by the same clients in all training rounds. We refer to these strategies as *variable* and *constant* client selection. The variable client selection strategy is consistent with the assumptions in the federated setting. In federated training clients may not be available during all training rounds, e.g., the device might be turned off. Our default client selection strategy considers this fact and allows different clients participate in different rounds of the training.

The image classification tasks becomes more complex when more data labels exist in the data. To capture this complexity a larger model might be required. The model required for datasets with more classes needs to have at least more neurons in the last layer of the deep learning network and hence has more parameters to learn. Images in the dataset we use in this study have two types of labels, a "coarse" label and a "fine" label. There are 20 different coarse labels and 100 different fine labels in the dataset we use. We use coarse labels as our default labels, but to observe the effect of the number of classes, in the last experiment, we train the models with coarse and fine labels and compare the performance of the trained models.

F. Experiment Process and Metrics

To evaluate the effect of each parameter on the federated model, we train the model using the data from participating clients. We report the training accuracy as a percentage of the correctly predicted labels by the model on the same data that

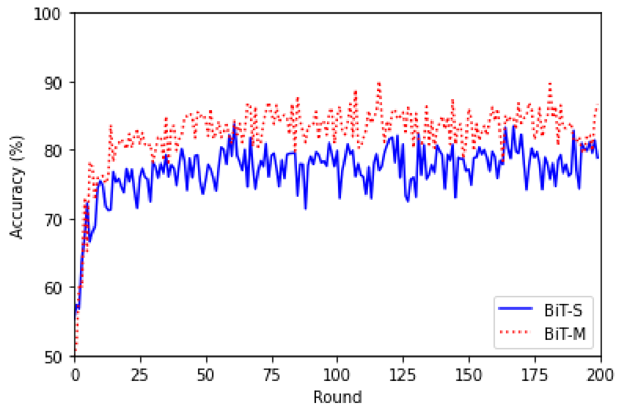
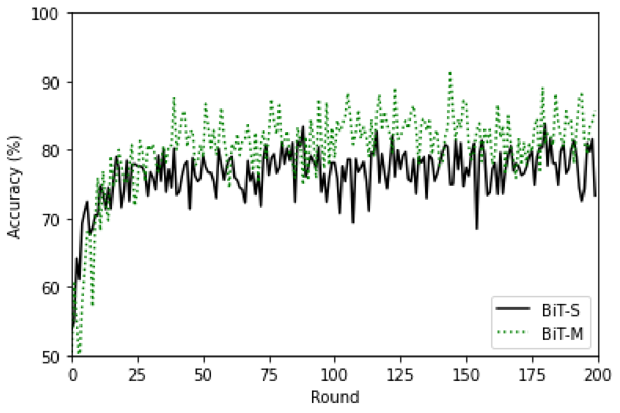
(1) $r50 \times 1$ models	(2) $r101 \times 1$ models
	

Fig. 4: Training accuracies using different base models

the model is trained on. We also report test accuracy, which is the accuracy of the model when predicting the labels of data records that the model has not seen during the training process. We repeat each experiment two times and report the average metrics.

V. Results

In this section we report the results of the experiments we have conducted to answer our two research questions. To answer the first question, we have built a federated model for image classification using transfer learning. We study the effect of the base model selection using four models in Section V-A. To study the effect of the federated parameters, we consider the effect of the training rounds and also the effect of clients participating in training models. We show these results in Section V-B. In Section V-C, we show the results of the experiment that studies the effect of the number of classes using two different base models and also two different client numbers.

A. The Effect of the Base Model

To compare the effect of the base model when training a federated deep learning network using transfer learning, we use four different models. As described earlier, these models are different in terms of their residual layers and also the source datasets used to train the models. Fig. 4 shows the training accuracy of all 4 models over 200 rounds of training. The left figure shows the accuracy of the $r-50 \times 1$ models for both S and M models, and the figure to the right shows the accuracy of the $r-101 \times 1$ models. In both cases, M models have a higher training accuracy. For example, the $M - r101 \times 1$ model returns an average accuracy of 71%, which is 13% higher than the average accuracy gained from $S - r101 \times 1$ model. This similar pattern is observed with $r50 \times 1$ models and emphasizes the importance of the source dataset that models are trained on. M models trained with a larger dataset have better accuracy regardless of the model sizes. To compare the effect of the size of the base models, we can compare the S models or M models across the two figures. S models achieve the same average accuracy of 58% as seen in the left and right graphs. M models achieve 67% and 71% accuracy under the same conditions. This shows that a larger model size is only useful when the source dataset is enough for training the model and learning more parameters that it has. When the source dataset size is small, the training accuracy will not get improved when a larger model is used.

To obtain a better understanding of the effect of the model size and the source data, we have plotted the test accuracies of all models in Fig. 5. Test accuracies are calculated every 25 rounds. Compared to the S models, M models achieve higher test accuracies. $M - r101 \times 1$ model achieves a higher overall accuracy than $M - r50 \times 1$ model, due to the higher number of learning parameters. We observe an opposite pattern in S models. $S - r101 \times 1$ has a lower accuracy than $S - r50 \times 1$ model. This can be explained due to the smaller size of the source dataset used for training S models. Models with more parameters to learn require a larger dataset to perform well on unseen test data. In S models, since the source dataset is small, it is not sufficient

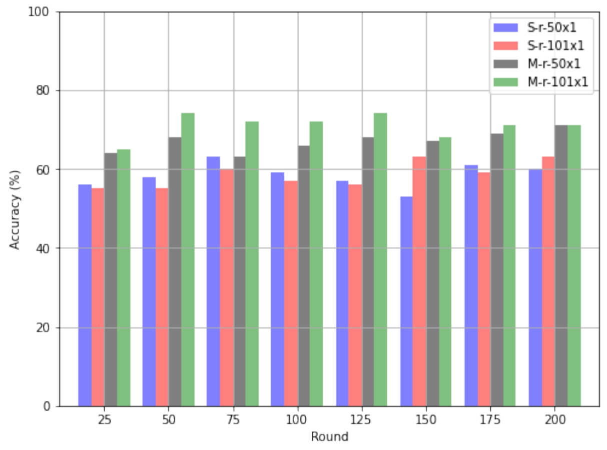


Fig. 5: Test accuracies with different base models

(1) Training Accuracy	(2) Test Accuracy

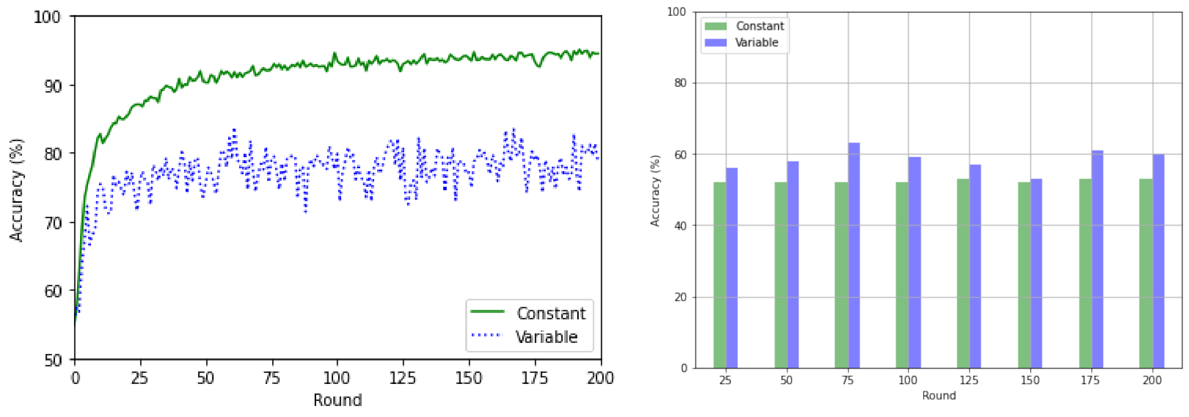


Fig. 6: Training and test accuracies with constant and variable clients

for a larger model, e.g., $S - r101 \times 1$ to be trained well enough to generalize for test dataset.

B. *The Effect of the Participating Clients*

As discussed earlier, a subset of all clients participate in the federated training process. The participating clients may not always be available to participate in all training rounds. For example, a client's device might be powered off during a training round and may not be available to complete the local model training in a single training round. Hence, clients participating in the federated training process may vary over training rounds. To observe the effect of the varying clients and also to compare the model accuracy under this case the same clients participate in all training rounds and then we train the $M - r50 \times 1$ base model with these two different client selection strategies. Fig. 6 shows the training and test accuracies of these two strategies.

The left graph in Fig. 6 shows the training accuracy of the two client selection strategies we are focusing on. For better visibility the Y-axis is clipped and starts at 50% accuracy. In both cases, the training accuracy increases over rounds of training. The training accuracies with variable clients gets to 80% around round 200, this is not very high compared to the constant clients which achieves 95% accuracy. While the constant client selection strategy outperforms the variable client selection strategy during training, this high accuracy is due to overfitting the model for the training dataset and therefore the trained model cannot generalize well for unseen dataset. This can be seen by comparing the test accuracies in the right graph in Fig. 6. The variable client selection outperforms the constant strategy in all training rounds, and on average, the test accuracy with variable clients is 6% higher than constant clients. While in each round, both strategies allow the model to be trained from the training dataset of 10 clients, the variable strategy allows the model to see datasets from a larger variety of clients during different rounds of the training. Clients may change over training rounds, and this prevents the model with variable clients from overfitting the dataset of clients used for training. This experiment shows that the variable client selection strategy is more effective. Moreover, it shows that the unavailability of some federated clients does not affect the training of federated models negatively, as long as, there are enough clients during training rounds to train the model. It worth noting that the base model used in this experiment is our default $M - r50 \times 1$ model. A larger model such as $M - r101 \times 1$ trained with variable clients can result in a higher accuracy.

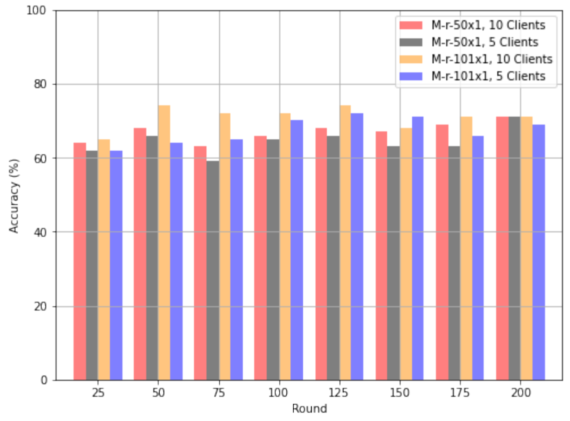


Fig. 7: BiT-M base models trained with 5 and 10 clients

Client selection strategy shows a higher effect compared to the number of clients participated in the training. Fig. 8 shows the training accuracy of both $M - r50 \times 1$ and $M - r101 \times 1$ models with 5 and 10 clients. In both models, the training accuracies with 5 clients follow the training accuracies with 10 clients very closely. Compared to the models trained with 5 clients, the test accuracies in Fig. 7 only shows a 3% and 4% improvement with 10 clients for $M - r50 \times 1$ and $M - r101 \times 1$ models. This improvement is very small considering that moving from 5 to 10 clients almost doubles the network communications for exchanging the model parameters with the federated server.

(1) $M - r50 \times 1$ model	(2) $M - r101 \times 1$ model

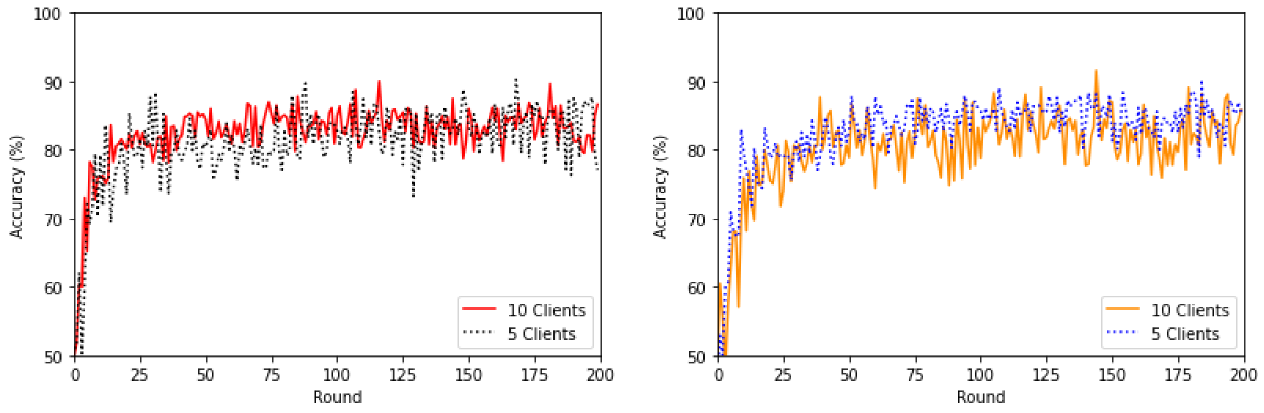


Fig. 8: The effect of the number of clients participated in training federated models

C. *The Effect of the Number of Classes*

In the previous experiments, we study the effect of parameters such as the number of client and the base model selection. These parameters can be chosen by data scientists and machine learning engineering based on the available resources or the trade-offs they offer. For example, a larger base model can be selected if the distributed edge devices have enough power to handle the model, or more clients can be used when the network overhead is acceptable.

In a classification problem, the more labels a dataset possesses the more complex the learning task becomes. The number of labels in a dataset is not a property that a data scientists can alter. Instead, in such more complex tasks, other parameters need to be adjusted to compensate for the loss of the accuracy the model can experience due to the difficulty of the learning task. In this experiment, we study the effect that the number of labels have on the model training and test accuracies. As explained earlier, each data point in CIFAR-100 has a fine and a coarse label. In total, the dataset has 100 fine labels and 20 coarse labels.

Fig. 9 shows the training and test accuracy of our default base model $M - r50 \times 1$ with 10 clients when trained with fine and coarse labels. The pattern in test and training accuracies remain consistent and the accuracies are lower for fine labels. With fine labels, while the model needs to learn 5 times more labels compared to 20 coarse labels, the accuracy only drops by 6%. This small drop in test accuracy compared to the substantially larger amount of labels that the model needs to learn shows small sensitivity of the models to the number of labels. To compensate for this accuracy drop the number of clients can be increased.

VI. Conclusions and Future Work

Federated learning allows distributed users to collaboratively train a model without sharing their private data. However, training large and complex models might be difficult due to the small size of data that each client owns. To overcome this challenge we have used transfer learning to build a federated model. The learning of base models trained on other datasets is used as the basis for models to be trained locally at clients. We have studied the effect of the size and

(1) Training Accuracy	(2) Test Accuracy

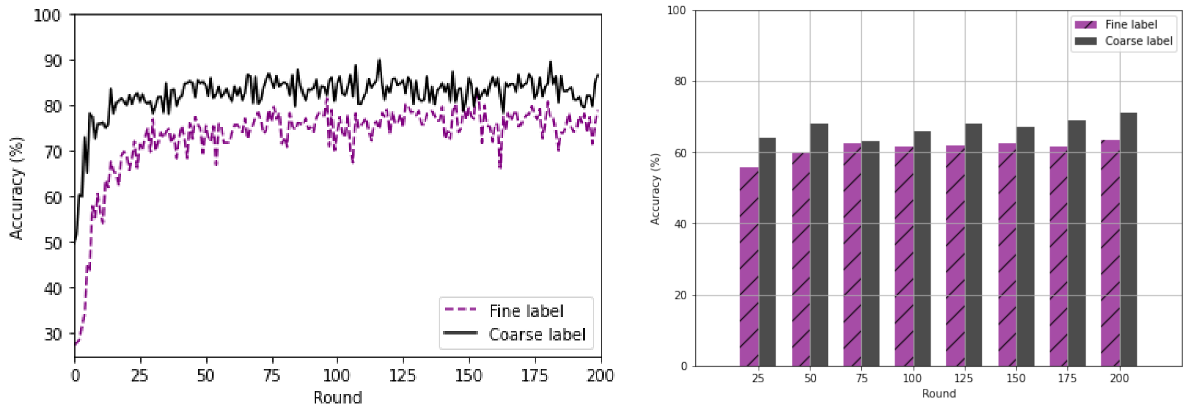


Fig. 9: Training and test accuracies with fine and coarse labels using the $M - r50 \times 1$ model

architecture of the used base models. The results show that transfer learning for training a base model is required when dealing with complex problems. We also observed that a larger model size should only be used when a larger dataset is available to train the base model. Moreover, the client selection strategy and the model size have high effects on the model accuracy. In the future, we plan to expand our analysis by training more models and considering device heterogeneity for client devices.

REFERENCES

[1] "General data protection regulations." https://gdpr.eu. Accessed: 2021-07-10.

[2] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[3] "Cifar-100 dataset." https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2020-06-10.

[4] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[5] Y. Liu, J. James, J. Kang, D. Niyato, and S. Zhang, "Privacy-preserving traffic flow prediction: A federated learning approach," *IEEE Internet of Things Journal*, 2020.

[6] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Federated learning for ultra-reliable low-latency v2v communications," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, IEEE, 2018.

[7] E. Bakopoulou, B. Tillman, and A. Markopoulou, "A federated learning approach for mobile packet classification," *arXiv preprint arXiv:1907.13113*, 2019.

[8] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, M. D. Mueck, and S. Srikanteswara, "Energy demand prediction with federated learning for electric vehicle networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[9] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 756–767, IEEE, 2019.

[10] A. Abeshu and N. Chilamkurti, "Deep learning: the frontier for distributed attack detection in fog-to-things computing," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 169–175, 2018.

[11] T. C. T. F. Authors, "Nvidia clara." https://developer.nvidia.com/clara, 2019. Accessed: 2020-07-20.

[12] T. P. Authors, "Paddlefl." https://github.com/PaddlePaddle/PaddleFL, 2019. Accessed: 2020-07-20.

[13] T. F. Authors, "Federated ai technology enabler." https://www.fedai.org/, 2019. Accessed: 2020-07-20.

[14] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[15] V. Mugunthan, A. Peraire-Bueno, and L. Kagal, "Privacyfl: A simulator for privacy-preserving and secure federated learning," *arXiv preprint arXiv:2002.08423*, 2020.

[16] S. Caldas, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[17] Y. Amannejad, "Building and evaluating federated models for edge computing," in *Proceedings of the International Conference on Network and Service Management (CNSM 2020)*, IEEE, in-press.

[18] "Tensorflow federated learning." https://github.com/tensorflow/federated. Accessed: 2020-06-30.

[19] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, pp. 1–8, 2018.

[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

[21] Y. Wang, "Co-op: Cooperative machine learning from mobile devices," 2017.

[22] G. Damaskinos, R. Guerraoui, A.-M. Kermarrec, V. Nitu, R. Patra, and F. Taiani, "Fleet: Online federated learning via staleness awareness and performance prediction," *arXiv preprint arXiv:2006.07273*, 2020.

[23] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2019.

[24] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 954–964, IEEE, 2019.

[25] T. Tuor, S. Wang, B. J. Ko, C. Liu, and K. K. Leung, "Overcoming noisy and irrelevant data in federated learning,"

[26] C. Wang and S. Mahadevan, "Heterogeneous domain adaptation using manifold alignment," in *Twenty-second international joint conference on artificial intelligence*, 2011.

[27] P. Prettenhofer and B. Stein, "Cross-language text classification using structural correspondence learning," in *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp. 1118–1127, 2010.

[28] J. T. Zhou, S. J. Pan, I. W. Tsang, and Y. Yan, "Hybrid heterogeneous transfer learning through deep learning," in *Twenty-eighth AAAI conference on artificial intelligence*, 2014.

[29] J. T. Zhou, I. W. Tsang, S. J. Pan, and M. Tan, "Heterogeneous domain adaptation for multiple classes," in *Artificial intelligence and statistics*, pp. 1095–1103, PMLR, 2014.

[30] L. Duan, D. Xu, and I. Tsang, "Learning with augmented features for heterogeneous domain adaptation," *arXiv preprint arXiv:1206.4660*, 2012.

[31] B. Kulis, K. Saenko, and T. Darrell, "What you saw is not what you get: Domain adaptation using asymmetric kernel transforms," in *CVPR 2011*, pp. 1785–1792, IEEE, 2011.

[32] Y. Zhu, Y. Chen, Z. Lu, S. J. Pan, G.-R. Xue, Y. Yu, and Q. Yang, "Heterogeneous transfer learning for image classification," in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[33] M. Harel and S. Mannor, "Learning from multiple outlooks," *arXiv preprint arXiv:1005.0027*, 2010.

[34] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 874–896, 2017.

[35] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pp. 491–507, Springer, 2020.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[37] "Tff simulation datasets cifar100." https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/cifar100/load_data. Accessed: 2021-06-10.

[38] "Google colab." https://colab.research.google.com/. Accessed: 2020-07-10.

[39] “Keras.” https://keras.io/. Accessed: 2020-07-10.

[40] “Federated Learning for CIFAR-100.” https://github.com/Yasaman-A/Federated-CIFAR. Accessed: 2021-07-31.