

Rowan University

Rowan Digital Works

Theses and Dissertations

10-25-2022

UTILIZING FEDERATED LEARNING AND META LEARNING FOR FEW-SHOT LEARNING ON EDGE DEVICES

Kousalya Soumya Lahari Voleti
Rowan University

Follow this and additional works at: <https://rdw.rowan.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Voleti, Kousalya Soumya Lahari, "UTILIZING FEDERATED LEARNING AND META LEARNING FOR FEW-SHOT LEARNING ON EDGE DEVICES" (2022). *Theses and Dissertations*. 3069.
<https://rdw.rowan.edu/etd/3069>

This Thesis is brought to you for free and open access by Rowan Digital Works. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Rowan Digital Works. For more information, please contact graduateresearch@rowan.edu.

**UTILIZING FEDERATED LEARNING AND META LEARNING FOR FEW-
SHOT LEARNING ON EDGE DEVICES**

by

Kousalya Soumya Lahari Voleti

A Thesis

Submitted to the
Department of Computer Science
College of Science and Mathematics
In partial fulfillment of the requirement
For the degree of
Master of Science in Computer Science
at
Rowan University
September 2, 2022

Thesis Chair: Shen Shyang Ho, Ph.D., Associate Professor, Department of
Computer Science

Committee Members:

Anthony Brietzman, Ph.D., Associate Professor, Department of Computer Science
Nancy Tinkham, Ph.D., Assistant Professor, Department of Computer Science
Ning Wang Ph.D., Assistant Professor, Department of Computer Science

© 2022 Kousalya Soumya Lahari Voleti

Dedications

I sincerely dedicate this work to my thesis advisor Dr. Shen Shyang Ho and my family, without you this would not be possible.

Acknowledgements

I express my deep sense of gratitude to my thesis advisor Dr. Shen Shyang Ho and shall remain grateful for his inspiring guidance throughout the project and for valuable suggestions in various phases of project work.

A special thanks to Dr. Anthony Breitzman, Dr. Nancy Tinkham, Dr. Ning Wang for serving as my thesis committee members and their help for finishing the thesis process.

I gratefully acknowledge the emotional support, encouragement and patience given by my parents. Without their understanding, this would not have been possible. Also, I would also like to thank my friends who were constantly supporting me through this journey.

Abstract

Kousalya Soumya Lahari Voleti
UTILIZING FEDERATED LEARNING AND MET LEARNING
FOR FEW-SHORT LEARNING ON EDGE DEVICES
2021-2022
Shen-Shyang Ho, Ph.D.
Master of Science in Computer Science

The efficient and effective handling of few-shot learning tasks on mobile devices is challenging due to the small training set issue and the physical limitations in power and computational resources on these devices. In this thesis, we propose a solution that combines federated learning and meta-learning to handle independent few-shot learning tasks on multiple devices (or clients) and the server. In particular, we utilize the Prototypical Networks to perform meta-learning on all devices to learn multiple independent few-shot learning models and to combine the models in a centralized data distributed architecture using federated learning which can be reused by the clients subsequently. We perform extensive experiments to (1) compare three different federated learning approaches, namely Federated Averaging (FedAvg), Federated Proximal (FedProx), and Federated Personalization (FedPer) on our proposed framework, and (2) explore the effect of data heterogeneity issue on the few-shot learning performance. Our empirical results show that our proposed approach is feasible and is able to improve the devices' individual prediction performance and improve significantly on the global model (on the server) using any of the federated learning approaches when the few-shot learning tasks are on the same datasets. However, the data heterogeneity problem still affects the prediction performance of our proposed solution no matter which federated learning approach we used.

Table of Contents

Abstract	v
List of Figures	viii
List of Tables	ix
Chapter 1: Introduction	1
Chapter 2: Background	5
2.1 Few-Shot Learning.....	5
2.2 Meta Learning for Few-Shot Learning	7
2.3 Federated Learning	9
2.4 Meta-Learning Using Prototypical Networks for Few-Shot Learning	11
2.5 Problem Setting and Study Objectives	12
Chapter 3: Related Work and Literature Review	13
3.1 Few-Shot Learning Using Meta Learning	13
3.2 Federated Learning	15
Chapter 4: Federated Few-Shot Learning Using Prototypical Network.....	18
4.1 Solution Implementation Overview	18
4.2 Experimental Scenarios	19
4.3 Implementation of the Proposed Solution on Experimental Scenarios	20
4.4 Algorithm Design of FedPer and FedProx.....	22
Chapter 5: Empirical Results	26
5.1 Dataset Descriptions	26
5.2 Dataset Preprocessing.....	27
5.3 Experimental Setups	27

Table of Contents (Continued)

5.4 Performance Measures.....	28
5.5 Experimental Designs and Implementation	29
5.6 Experimental Results and Discussions	29
5.6.1 Performance Comparison of 2 Layer Ratios in FedPer Using Fashion-MNIST	29
5.6.2 Effect of FedProx Proximal Term on Prediction Performance Using Fashion-MNIST	31
5.6.3 Performance Comparison of Proposed Solution Using Different Federated Learning on Each Dataset in Single Dataset Scenario	32
5.6.4 Performance Comparison of Proposed Solution Using Different Federated Learning on Each Dataset in Single Dataset Scenario	42
Chapter 6: Conclusions and Future Work.....	51
References.....	53

List of Figures

Figure	Page
Figure 1. Overview of Federated Few-Shot Learning Using Meta Learning.....	2
Figure 2. Implementation of Proposed Solution.....	20
Figure 3. Server Side Global Model Pre-Training.....	22
Figure 4. Sample 42 Base Layers for FedPer	24
Figure 5. Prediction Performance as μ Varies on Fashion-MNIST (2 Clients)	31
Figure 6. Fashion MNIST Single-Data Results	35
Figure 7. CIFAR-100 Single-Data Results.....	39
Figure 8. Omniglot Single-Data Results.....	42
Figure 9. Comparison of Fashion-MNIST on Single-Data and Multiple-Data Scenarios	48
Figure 10. Comparison of CIFAR-100 on Single-Data and Multiple-Data Scenarios.....	49
Figure 11. Comparison of Omniglot on Single-Data and Multiple-Data Scenarios.....	50

List of Tables

Table	Page
Table 1. Base+Personalization Layers Effect on Fashion-MNIST (2 Clients).....	30
Table 2. Results on Fashion-MNIST Dataset	33
Table 3. Results on CIFAR-100 Dataset.....	37
Table 4. Results on Omniglot Dataset	41
Table 5. Results of Multiple-Data Scenarios	43

Chapter 1

Introduction

There is a rapid growth in mobile device usage over the last decade. Moreover, there is a need to build effective predictive models on these mobile devices for different user needs. In other words, predictive models are different on different devices. The main challenge to build these predictive models is the limited amount of data for each object class (e.g., five to ten images available for each class) available for the predictive model on a device. This is the so-called *few-shot learning* problem [1].

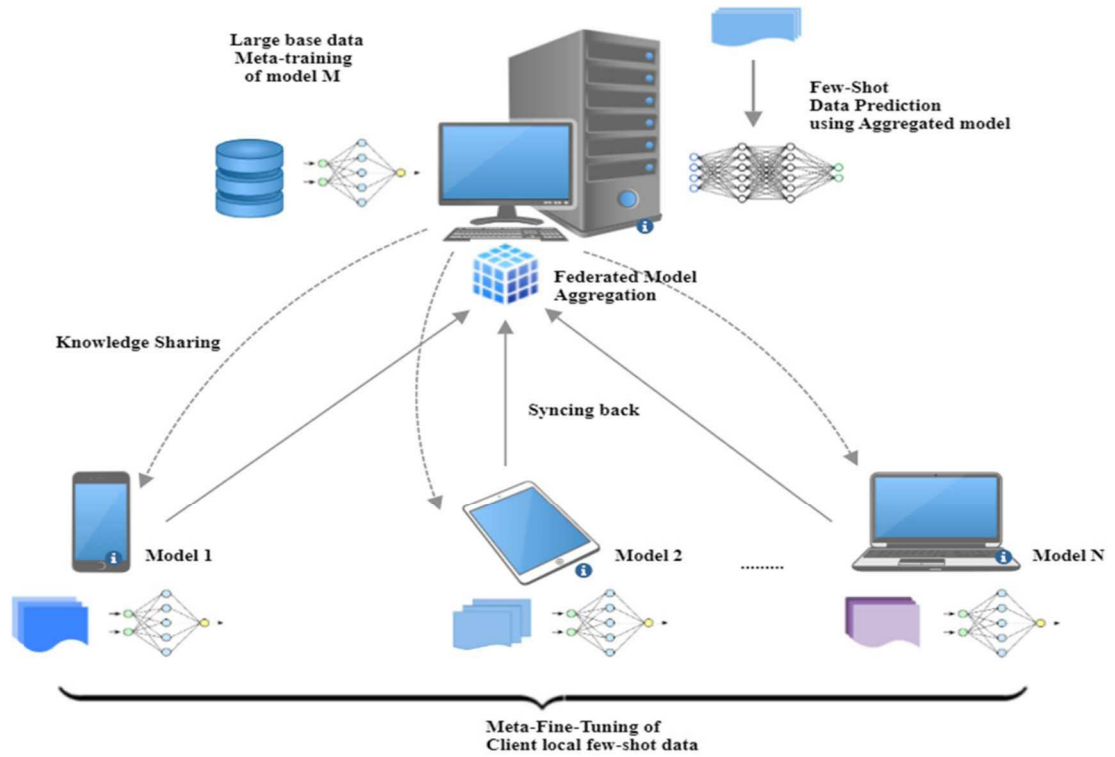
Federated Learning (FL) [2, 3] is an evolving technique which can solve the few-shots learning issue [1] by allowing the edge devices to collaboratively train and share knowledge to improve the prediction accuracy at each device. In particular, distributed devices can effectively train their models and aggregate them to form an effective global model shared by the devices.

The key difference between our few-shot learning scenario and scenarios on existing federated few-shots learning problems is that each device has its own distinct prediction task different from the others. Moreover, we have a centralized server that has an initial larger, but non-overlapping dataset as compared to the data available in the devices. Our proposed solution combines federated learning (using aggregated models trained for the few-shots learning tasks) with *meta-learning* [4] to fine tune (e.g., the distance metrics and parameters for the) predictive models (at the devices) so that they work well when the number of data samples for each class is limited on new few-shot learning tasks at the mobile devices. Figure 1 shows an example of our problem scenario and a high-level sketch

of the proposed solution utilizing federated learning for knowledge sharing (models) among multiple devices to perform few-shot learning at these devices driven by meta-learning to fine-tuning the individual models.

Figure 1

Overview of Federated Few-Shot Learning Using Meta Learning



To enhance the efficiency of few-shot learning on the devices, a meta-learning technique is applied on each device on a collection of few-shot learning tasks so that global and local

predictive models can be efficiently learned for unseen few-shot learning tasks on both the server and devices. Many meta-learning methods have been presented for the few-shot learning problem such as Task Agnostic Meta-Learning [5] and meta-learning over a pre-trained model on the whole datasets using some evaluation metric [6]. In this thesis, we utilize Prototypical Networks [7] to fine-tune the predictive model by learning the metric space that the few-shot classification tasks can be performed the best based on the data available on the mobile device. Unlike meta-learning approaches such as MAML, PT-MAP [8, 29] which includes higher order derivatives, computationally expensive algorithms and has longer run times, Prototypical Networks instead is computationally efficient and provides much more stability.

In this thesis, we implement our proposed federated few-shot learning framework utilizing Prototypical Networks to perform meta-learning on all devices in a centralized data distributed architecture such that different few-shot predictive models can be executed on the devices and the server. We perform extensive experiments on three real-world datasets, namely CIFAR-100 [9], Fashion-MNIST [10] and Omniglot [11], to (1) compare three different federated learning approaches, namely Federated Averaging (FedAvg) [12], Federated Proximal (FedProx) [13], and Federated Personalization (FedPer) [14] on our proposed framework and (2) explore the effect of data heterogeneity (using different datasets on different edge devices) on the few-shot learning performance. The main observations and conclusions from our empirical results are as follows:

1. Varying the FedProx proximal term μ between 0.01 and 1.5 does not have a significant effect on the prediction performance for our proposed approach using FedProx for federated learning.

2. For few-shot classification tasks with reasonable difficulty ($> 50\%$ accuracy), the proposed approach is able to improve the devices' individual prediction performance and improve significantly on the global model (on the server) using any of the federated learning approaches when the few-shot learning tasks are from the same datasets.
3. Unsurprisingly, the aggregated (global) models from FedPer perform the best most frequently, followed by aggregated models from FedProx.
4. Data heterogeneity problem affects the prediction performance of our proposed solution no matter which federated learning approach we used.

The thesis is organized as follows. In Chapter 2, we describe the few-shot learning problem, how meta-learning approaches are used to build few-shot learning solutions, and the federated learning setting. Then, we describe our problem setting in detail. In Chapter 3, we describe previous work on meta-learning methods, and their use to build few-shot learning solutions, federated learning, and meta-federated learning. The proposed federated-learning-driven few-shot learning solution using meta-learning, the experimental scenarios, its implementation, and related issues are described and discussed in detail in Chapter 4. In Chapter 5, we present extensive experimental results to study the proposed Prototypical Network based solution for few-shot learning tasks on devices using three different datasets and three different federated learning approaches. Chapter 6 is our thesis conclusions including possible future work.

Chapter 2

Background

In this chapter, we describe the few-shot learning problem, and how meta-learning techniques have been used to handle few-shot learning problems. Then, we describe the two different federated learning architectures. Moreover, we introduce Prototypical Networks which we use for meta-learning of few-shot learning tasks in a federated learning setting. Finally, we provide a description of the centralized architecture for the federated learning setting considered in this thesis and the assumptions on the data available on the server and the edge devices.

2.1 Few-Shot Learning

Few-shot learning (FSL) [1] is a machine learning task which particularly deals with developing models which can best predict on a limited amount of data. It is a learning problem given only a few examples per class. During multi-class classification, when there is only one example per class, FSL is termed as *one-shot learning* [15] and if there is no example per class, then it is termed as *zero-shot learning* [16]. Driven by the goal of making machine learning models a better predictor, more human-like and less computationally expensive, research on few-shot learning has been a recent hot topic. Since the model is being deprived of data, traditional supervised learning methodologies cannot handle the problem effectively to ensure good predictive performance. To address this problem, new approaches driven by other machine learning solutions such as meta learning [17], multi-task learning [18], adversarial learning [19], generative modeling [20] have been proposed to overcome the challenges in few-shot learning tasks.

Multi-task learning [18] uses *parameter sharing* as a technique to solve FSL. For this learning problem, there exists a set of multiple related tasks for the model to learn and predict. All these tasks consist of classes with fewer samples (few-shot tasks) and with a large number of samples (base tasks). The few-shot tasks are used for fine-tuning training, whereas the base-tasks are considered as prior knowledge for pre-training. Each task is first divided into a training set and a test set for the model to be trained individually. After initial training of the model using the training set of base-tasks, the feature extractor layers of this model are divided into task-generic layers (to share their parameters with model training on other tasks), and task-specific layers (to specialize current tasks). The model trained in this way is further fine-tuned on a training set of few-shot tasks and finally, one can validate the model using the test set of few-shot tasks.

Another few-shot learning approach is based on adversarial learning and generative modeling techniques of machine learning [20]. These are broadly termed as *Hallucination based algorithms*. These algorithms directly deal with the data deficiency by data augmentation. The basic assumption of these techniques is that a model can learn some intraclass relationships (e.g., variance, etc) during training of given samples, which can be further applied to a new few-shot learning task. One recently proposed method is the Adversarial Feature Hallucination Network (AFHN) [20], which is a GAN-based algorithm for solving FSL. It learns the image feature representations of given classes and synthesizes them using a conditional framework and uses this knowledge for few-shot classification. In other words, they perform data augmentation (from learned knowledge) for the limited number of samples from which they can hallucinate and predict on unseen data.

2.2 Meta Learning for Few-Shot Learning

Meta-learning [4] is learning from one model with a large amount of data, and fine-tuning it to another model for a new similar task. One recent research of interest is whether one can utilize meta-learning techniques to support *few-shot learning* tasks [1]. Unlike supervised learning of training on one set and testing on another set, meta-learning technique uses three different sets, namely: **base set** for prior training, **support set** for fine-tuning and a **query set** which one performs prediction on. The support set and query set mostly have classes with fewer samples that are unseen in the base set.

Every support set and query set is specified as **n-way: k-shot: q-query** tasks where *n-way* denotes number of classes in the sets, *k-shot* is number of images per class in support set and *q-query* is the number of images per class in the query set. In this section, we describe how meta-learning can be used to solve the few-shot learning problem.

The meta-learning solution is broadly classified into two categories:

1. Inductive Approach (Supervised): Meta-learning methods follow traditional supervised learning techniques in which the prediction is made on totally unseen samples. The trained generalized model is used to make a prediction. The two main steps of such meta-learning methods are as follows:

- 1) **Creating a set of Support and Query sets for Training.** To learn to handle an unseen few-shot learning task, a training set is needed to build prior knowledge of the model. This training set is also called a base set containing a large number of classes and images in each class. The base set is randomly sampled to create multiple support and query sets of pre-defined n-way k-shot q-query configuration for n-way k-shot few-shot learning training purposes. By replicating the process

of predicting with support and query sets during training, when the model needs to train on a new unseen support and query set, the training will be more efficient and effective.

- 2) **Fine Tuning Using a Support set.** The pre-trained model is given a support set (or a **novel set**) of n-way k-shot configuration. One will fine-tune the predictive model (based on the base set) for the query set using the support set.

A **query set** (or a **validation set**) with n-way q-query is used to test the performance of the fine-tuned prediction model. In other words, a prediction model which is trained on a large base set and fine-tuned on an n-way k-shot support set will predict on an n-way q-query query set.

An important point to note here is that for most cases the n-way k-shot q-query set can be just termed as n-way k-shot which means that both the support set and query sets are different but use the same quantity of images. In this thesis, we referred to it as *n-way k-shot q-query* in our experimental results. Every client and the server chooses random support and query sets from their data loaders which will be clearly explained in Chapter 5.

Some of the popular existing meta-learning methods which follow the above steps are Prototypical Networks [7], Matching Networks [31], Relation Networks [32], MAML (Model Agnostic Meta-learning) [8] etc. These algorithms are explained in detail in Chapter 3. Apart from these, there is also another category of algorithms called *Transfer Learning Baselines* [17] which follow very much similar steps as meta-learning approaches but are slightly different in the usage of the small number of data from each class. These algorithms *do not follow episodic learning* but instead transfer relevant

knowledge from the trained model to the particular few-shot learning tasks by undergoing fine tuning. Examples of this type are Baseline and Baseline++ [17].

2. Transductive Approach (Semi-Supervised): Such an approach is limited to the performance of the current task and not generalized to multiple tasks. In this case the classifier will have access to both support set and query set making it very easy for the model to predict with greater accuracy than compared to the inductive approach. In fact, the approach has only one single step, in which the transductive classifier takes both the training samples which are labeled as well as unknown samples and makes a prediction on these unknown samples. Since the prediction is made on a batch of unlabelled data inputs that is already known, and these training samples are not provided with labels, it is thus called *semi-supervised meta-learning or transfer learning approach* [28]. There have been a number of methods and algorithms explored in recent times for this setting in the field of meta-learning [27, 33 and 34]. One example of transductive approach is PT-MAP (Power Transform- Maximum A-Posterior) [29] method which uses Gaussian like Distributions for handling semi-supervised data (See Section 3.1).

2.3 Federated Learning

Federated learning techniques allow one to train predictive model(s) across multiple edge devices or servers that have their own local datasets, without sharing the data explicitly. In a federated learning problem setting, it is no longer like the traditional centralized (i.e, single location) technique which builds a single predictive model using datasets from different devices on a single server and often with the assumption the local datasets on the edge devices are identically distributed. This learning setting addresses data privacy, data security, data access rights and access to heterogeneous data in many real-world

Internet-of-Things (IoT) applications [2].

In a federated learning scenario, individual edge device (aka client) participation in this case will be highly beneficial to both the clients and the server. The process of federated learning relies on the regular communication between the clients and/or with the server (if any). After completion of each communication round, an aggregation method is used to combine models from the clients to improve the predictive model. The data provided by the clients can be of two types: Independent and Identically Distributed (IID) or Non-IID [13]. Dealing with IID data is probably the easiest task for any federated learning related problem but the real challenge is to build a robust predictive model using Non-IID data.

The federated learning problem setting can be characterized into two ways:

1. *Centralized Architecture*: The server creates a global model, sends it to the clients for getting trained on their own private data and then they just send back the model parameters to be aggregated at the server. The flow of data is asymmetric and communication between the edge devices can be synchronous or asynchronous. A simple aggregation approach for this architecture is FedAvg [12] which takes the average of the model parameter values from the clients.
2. *Decentralized Architecture*: There is no requirement of a trusted server or manager device. In each communication round, one client updates the local parameters of their local data, then selects another party and sends its computed gradients or model parameters to the selected client(s). Next, the party which is selected uses these parameters to train a local model using its own private data. The selection and sending process continues until all parties finish in one round.

Every client is selected for updating the global model for the same number of rounds. An example of the federated learning algorithm for this architecture is SimFL [30].

2.4 Meta-Learning Using Prototypical Networks for Few-Shot Learning

In this thesis, we explore the utilization of *Prototypical Networks* [7], an efficient meta-learning algorithm which prioritizes the few-shot learning in the federated learning setting. These networks are characterized by calculating the prototypes (i.e., centroid of a feature space). Consider a support set S of n labeled examples from k classes $\{c_1, c_2, \dots, c_k\}$ given by $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ where $\{x_1, x_2, \dots, x_n\}$ are input images and $\{y_1, y_2, \dots, y_n\}$ are their respective labels. Let S_k denote all the examples of class k and we use its total number of examples for calculating the mean [7]. We pass these images to the feature extractor parameterised by function F and get feature vectors $\{z_1, z_2, \dots, z_n\}$. For each class c_k , we calculate the average of feature vectors, and find k prototypes $\{p_1, p_2, \dots, p_k\}$. Mathematically, [7]

$$p_k = \frac{1}{|S_k|} \sum_{(x_i, y_i)} F(x_i) \quad \text{where, } (x_i, y_i) \in S_k \quad (1)$$

Now when a query set image, say x_q is given, we again extract its feature vector using F to get z_q and calculate Euclidean distance between z_q and each prototype p_k . Since we have k prototypes we get k distances, $\{d_1, d_2, \dots, d_k\}$. Because we cannot predict using these distances, we calculate the negative logarithmic probability distribution of these distances using softmax. The class k with highest softmax values is predicted as an output class for x_q . The learning proceeds by reducing the loss using SGD optimization [36].

The Prototypical Networks, unlike other meta-learning methods, are computationally inexpensive as well as efficient. Since it is based on distance metric calculation of prototypes of each class, they are very easy to implement. Also estimation of these prototypes is done using mean calculation, which makes it noise resistant.

2.5 Problem Setting and Study Objectives

In this thesis, we consider a centralized architecture as described in Section 2.3 similar to the example in Figure 1 in Chapter 1. For our problem setting, the server has a larger training set (i.e., base set) at initialization for the meta-learning process. The clients (i.e. edge devices) have smaller non-overlapping sets (with similar classes present in training set) which are used to generate support sets for few-shot learning on an *n-way k-shot* prediction problem on *q-query* over 20 communication rounds.

The server performs a federated learning aggregation process at each round using the meta-learning models for few-shot tasks learned at the edge devices and shared with the clients. Our main investigation objectives are to understand

- 1) The performance behavior (on clients and server) of 3 different federated learning aggregation approaches described in Chapter 4 on our proposed meta-learning few-shot learning solution.
- 2) The performance and learning behavior over multiple communication rounds for our proposed solution.

Chapter 3

Related Work and Literature Review

In this chapter, we will review some important previous work related to few-shot learning using meta-learning, federated learning, and meta-learning to improve federated learning.

3.1 Few-Shot Learning Using Meta-Learning

- 1) **Matching Networks** [31]: This is a meta-learning algorithm which is very much similar to Prototypical Networks (see Section 2.4). The only difference is that the MatchingNet uses cosine distance as its distance metric instead of Euclidean distance. It calculates the average cosine distance for each class between the query feature and each support feature. The algorithm uses the learned embedding space for the support set and for the query set.
- 2) **Relation Networks** [32]: These neural networks divide the few-shot learning classification problem into two modules, an embedding module to retrieve the feature representations of the query set images and a relation module (a more deeper learnable comparator) instead of a standard linear one to compare the properties of image categories in the support set and query set [32]. Several different feature maps are extracted from the average of all the support set images from the embedded module to be fed into the relation module in the later stage to produce scalar range relation scores from 0 to 1 which represents the similarity or dissimilarity between the considered query image and support image.
- 3) **MAML** [8]: Model Agnostic Meta-Learning is a unique and a very powerful algorithm compared to all the others. Its main goal is to learn how to initialize

good parameters of the model which can successfully make an accurate prediction based on optimal minimization of the loss function. Here one does not consider any embedded feature vectors but rather the neural network is given the entire large training set (divided into episodes) to make a supervised learning prediction out of learned features. Then this network will be added with a linear layer that predicts the output. Next, the gradient information from the loss function is used to fine-tune the same neural network but on a specific set of support points until it can better predict query points. One important thing to note is that, it does not learn an update function or a learning rule, but it learns the model parameters in a differentiable way. The name *agnostic* means that it can be used in different task contexts. Because it has an ability to deal with different types of data and is able to make a good prediction on it, it is therefore used for meta-federated learning where data heterogeneity is a major concern.

Until now we have discussed the inductive methods; the following is a pretty good example of a transductive approach.

- 4) **PT-MAP Transductive** [29]: Unlike the traditional supervised learning, in the transductive approach some of the unlabelled query samples are given to the neural network during the training process. Power Transform- Maximum A Posterior algorithm (PT-MAP) [29] concentrates on applying preprocessing and transformation techniques for the feature vectors from the support set and query set to be more aligned to Gaussian-like Distributions (Power Transformation phase). These distributions then undergo a technique called Sinkhorn Mapping for a number of iterations (MAP phase). For each iteration, the class centers are

updated and after all iterations, the prediction is made on the query set.

3.2 Federated Learning

The conventional centralized federated learning algorithms involve the following two stages: First, one of the server trains a global model with a huge amount of data. Then it randomly chooses N clients among K clients and then sends them the server trained model for training on their local data. These selected clients send back their models to the server to be combined in a useful way. One of the oldest, basic yet frequently used federated learning approaches is Federated Averaging (**FedAvg**) [12] which basically averages the weights of all the client models and updates the server model. The main weakness for FedAvg is that it cannot handle data heterogeneity in the different clients. All the algorithms that have been developed after this traditional algorithm use the same aggregation idea but with different modifications to overcome its drawbacks. In order to handle data heterogeneity, Federated Matched Averaging (**FedMA**) [21] performs simple matching of models using Probabilistic Federated Neural Matching before averaging. **FedDist** [22], again based on model matching, combines FedAvg and FedMA. For FedDist, during model aggregation FedAvg is performed first and similarity between a client model and the aggregated model is based on Euclidean distance. Additional statistical information on the client models are utilized to decide whether a client model should be aggregated into the server model. Both FedMA and FedDist attempts to identify the client models which can diverge due to dissimilar and data heterogeneity and avoid their inclusion into the aggregated server model.

Another important work done in this field is the **FedRep** [23]. The intuition behind this is that all the data which has been learnt by the clients share a common feature

representation which can be used by the clients again to build a more personalized classifier which can further determine each client’s local data labels. In this thesis, we perform federated learning using the basic FedAvg, FedPer and FedProx algorithms (see Chapter 4) and compare their performances on few-shot learning tasks.

There have been attempts to integrate meta-learning into federated learning solutions. For example, **FedMeta** [24] and **Per-FedAvg** [25] are proposed to improve the FedAvg algorithm. Both algorithms used MAML (See Section 3.1) to improve FedAvg so as to deal with non-homogeneous data. FedMeta reduces the communication overhead with faster convergence of client performances and efficiently increasing the prediction performance. Per-FedAvg emphasizes on personalization of client local data usage using MAML and handles data heterogeneity. Fed-Meta has been tested only on LEAF [26] datasets. **FedFSL** [19] tackles the few-shot learning with federated learning and adversarial learning techniques. It creates a separate feature space for each client and uses adversarial learning techniques for prediction.

Three federated learning aggregation approaches to be integrated into our proposed solution framework and compared in our empirical experiments are described below:

- 1) **FedAvg** [12]: When the clients receive the server pre-trained model, they undergo several rounds of local training for their individual data. At each round, these clients update the weights and finish their local training, their weight updates are averaged and sent back to the server for further testing. The client and the server model use the Stochastic Gradient Descent (SGD) [36] optimization method for parameter updates to minimize the loss function. It is a simple averaging technique and it does not address the data heterogeneity issue among the clients.

- 2) ***FedPer*** [14]: Fed-Personalization as the name suggests, concentrates more on the individual learning process of the clients. The more the clients learn, the better is the overall aggregated global server model performance. So, the client neural network has its layers divided into base and personalization layers where base layers often get updated with the FedAvg aggregated model layers and personalization layers are kept aside for client specialization. For every communication round only the base layers are changing with respect to the server model but the personalization layers are never changed.
- 3) ***FedProx*** [13]: Federated Optimization (**FedProx**) specifically addresses and deals with the inconstant resource constraints of clients during federated learning and also the issue with heterogeneity of local data at the clients. They assure the non-uniform working of different client devices and give each client a varying amount of work to be done. They use a proximal term for this process, which balances the local updates.

We will see the algorithmic design of these algorithms in more detail in Section 4.4

Chapter 4

Federated Few-Shot Learning Using Prototypical Network

In this chapter, we describe our proposed methodology for federated few-shot learning driven by meta-learning using Prototypical Network on all clients in a centralized data distributed architecture such that different few-shot predictive models can be executed on the clients.

4.1 Solution Implementation Overview

Our solution implementation on a centralized architecture (see Section 2.3) with a server and multiple edge devices (See Chapter 1 Figure 1) consists of three main steps. First, we have a huge dataset on the server that allows us to perform *meta-learning for few-shot learning on the server* in an episodic manner. A prototypical neural network model is created using a set of few-shot learning tasks randomly generated from the huge dataset. Second, this *trained meta-learning model is sent to all the clients for training and to fine-tune this model* with their own local data which are smaller in size. Note that the data provided by the clients is always few-shot in a fixed n-way k-shot q-query configuration based on the problem of interest. Third, we perform the *aggregation process of federated learning*. In other words, after the completion of client local training, their models are sent to the server through an aggregation method. The three steps are iterated for multiple rounds in our experimental scenarios.

4.2 Experimental Scenarios

To explore the feasibility and efficacy of our few-shot solution using meta-learning and federated learning on both homogeneous and heterogeneous data, we perform experiments on two scenarios.

- 1) **Single Dataset:** The server and all (two or three) clients learn from the same dataset. The dataset of interest is divided among the server and clients. The server gets a huge portion of it and the rest of the dataset is divided equally among the clients.
- 2) **Multiple Datasets:** Server has two datasets and every client (three of them) has a different dataset. Aggregation may not be done on a particular client. The main objective is to test the effect of performing fine-tuning on few-shot learning models learned from the two datasets (on server) on an unrelated dataset (on a client).

We limit our experiments on the three aggregation algorithms to just three clients. We create a simulated centralized federated learning architecture for the server and three clients and since all the datasets are not in equal length, we consider either two or three clients in active mode depending on the dataset used. We also limit the number of shots in the few-shot learning tasks on the clients. We consider three few-shot learning task configurations, namely: (i) 3-way: 5-shot: 10 query, (ii) 5-way: 5-shot: 10-query and (iii) 5-way: 5-shot: 5-query. We have chosen these particular configurations so that we can make a fair comparison between the way-change and shot-change. The number of clients used is different for each dataset since three datasets (see Chapter 5) we used in our experiments are not of the same size. The feature extractor we used in our Prototypical

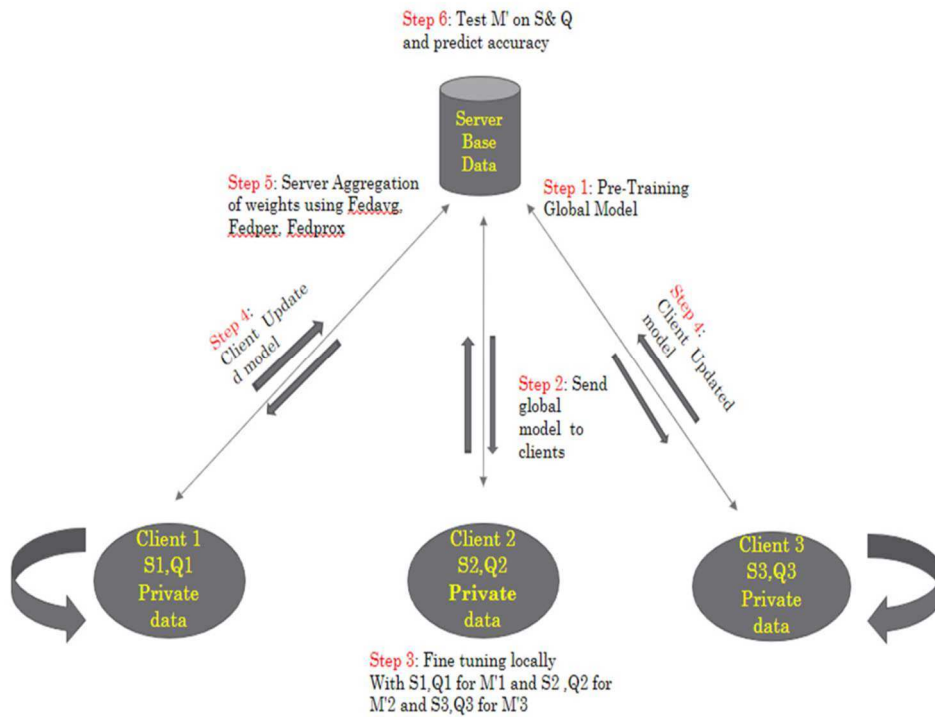
Networks is ResNet18, and SGD optimization is used during training.

4.3 Implementation of the Proposed Solution on Experimental Scenarios

The steps we followed for a single round of federated learning is shown in Figure 2. We assume three clients in our implementation description below. Every dataset is divided into five parts if it is a three-client scenario or four parts if it is a two-client scenario.

Figure 2

Implementation of Proposed Solution



Step 1: The dataset (B) with a huge number of data points is considered at the server side for base training, This dataset is further randomly sampled into support (S_b) and query

set (Q_b) of predefined few-shot configuration by the **global model** M which is based on Prototypical Network. Z_s is the feature space of the support set for which prototypes are calculated. Now using these prototypes, we calculate the Euclidean distances to query set feature space Z_Q (see Section 2.4). Finally, using softmax and fully connected layers we predict the label of each query datapoint and minimize the loss using SGD. This process happens for 400 such randomly sampled support + query sets which is called episodic training. In Figure 3, we see a detailed view of this.

Step 2: The global model M is sent to the clients.

Step 3: Depending on the number of clients, each client randomly chooses a support set and query set of a predefined few-shot learning configuration (e.g., 5-way, 5-shot, 5-query). Once the clients receive M , they perform model fine-tuning with their distinct support sets $S1, S2, S3$ and perform prediction on their query sets $Q1, Q2, Q3$ using their respective fine-tuned model $M'1, M'2, M'3$. Note that the client undergoes fine-tuning with just 1 support set and query set instead of episodic training.

Step 4: Local copy of global models which are updated in Step 3, are sent back to the server.

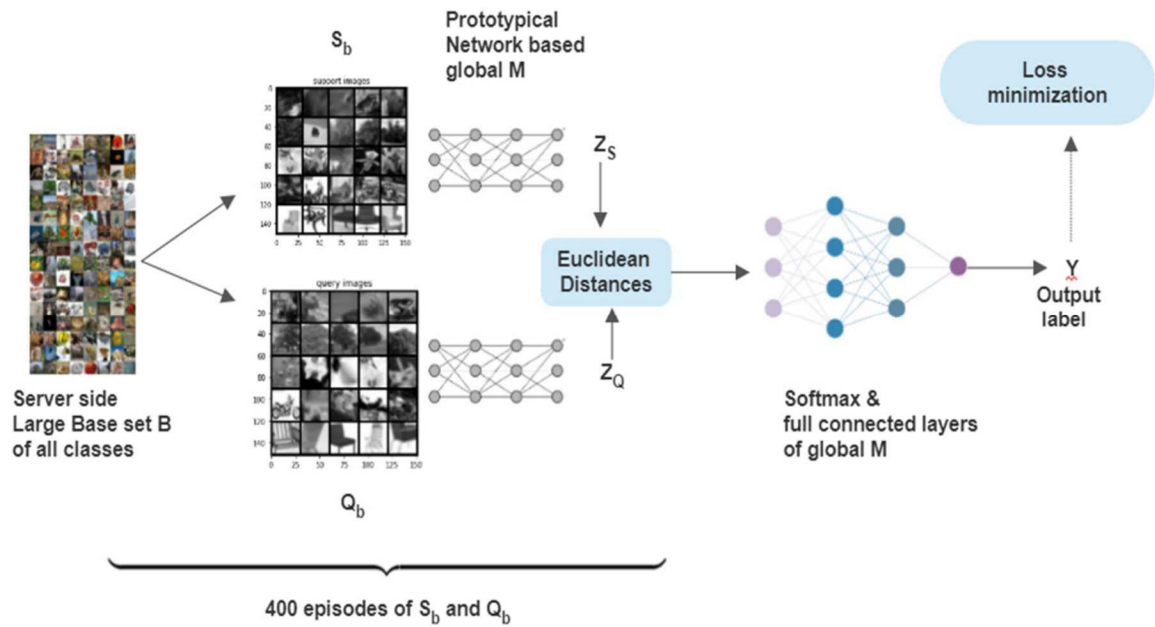
Step 5: Next, we perform model aggregation of the updated models on the server, using one of the federated learning algorithms (FedAvg, FedProx, and FedPer) described in Chapter 3. The resulting model is referred to as M' .

Step 6: Using M' we test the server using S and make a prediction on Q and obtain a server testing accuracy.

These steps are iterated for multiple rounds (20) in our experimental scenario implementation.

Figure 3

Server Side Global Model Pre-Training



4.4 Algorithm Design of FedPer and FedProx

FedPer: FedPer allows a client to learn a **local model** using the client's own local data to overcome the data heterogeneity issue in federated averaging (FedAvg). FedPer divides the client neural network layers into base layers and personalization layers where base layers are always updated with federated averaging method which happens for every round in the training, but the personalization layers remain unchanged. The intuition behind this

implementation is that these personalized layers can help capture the client's local data patterns.

For Prototypical Networks in our proposed solution, we use ResNet18 [35] as our feature extractor. ResNet18 has 1 residual block and 4 sequential layers, each consisting of 2 basic blocks which makes it a total of 9 blocks (residual + basic). A basic block consists of 2 convolutional 2D layers. Including activation function and bias layers, a residual block has 6 layers and each of the 8 basic blocks have a total number of layers in the order: {12,12,18,12,18,12,18,12}. Including the activation function layer and normalization layers, the ResNet18 architecture consists of 120 layers. We divide these layers into two configurations of base and personalization layers: (1) 78 base layers and 42 personalization layers, and (2) 42 base layers and 78 personalization layers.

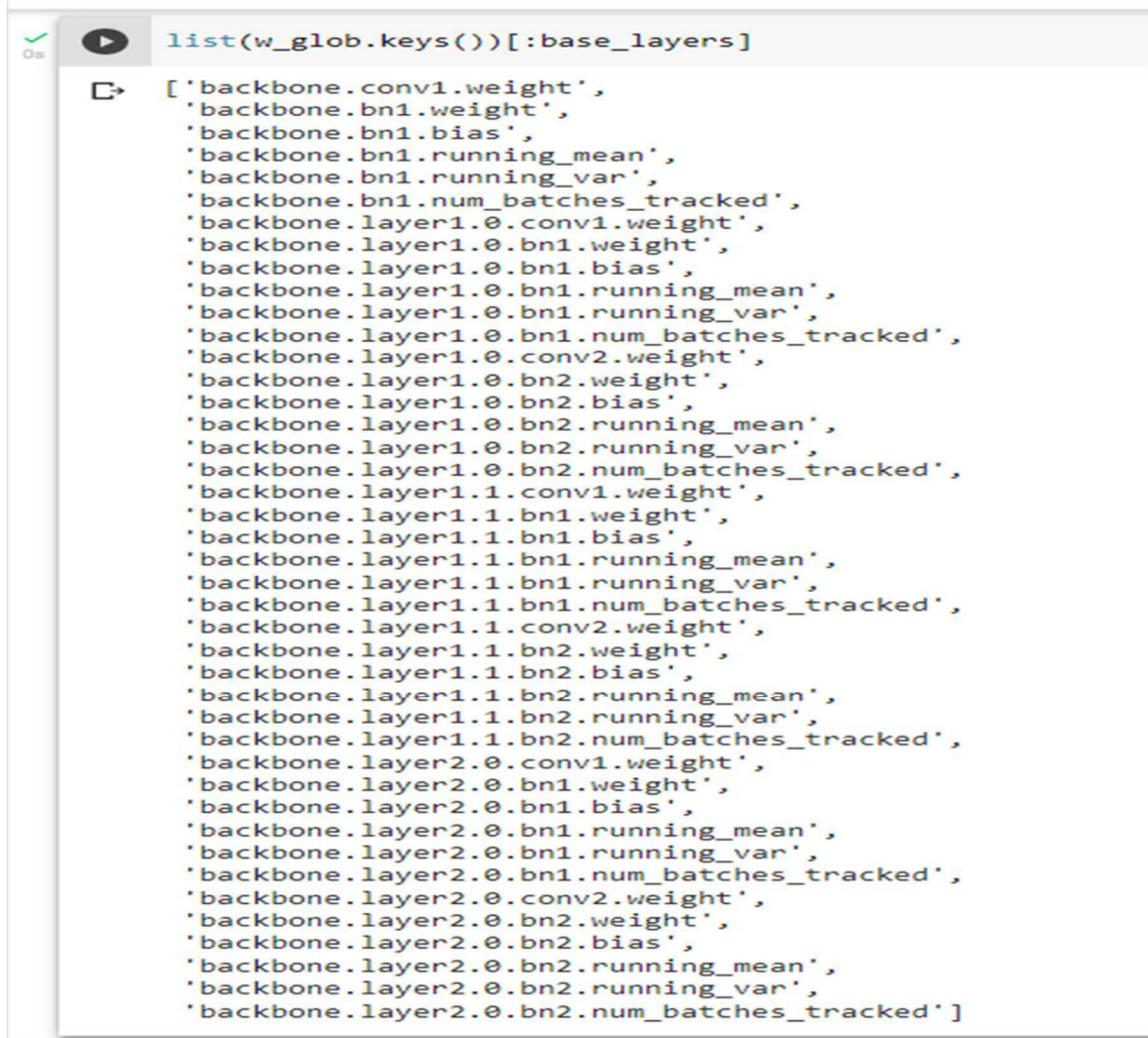
For the first configuration, the base layers consist of 1 residual block (6 layers) along with the first 5 basic blocks (72 layers) and personalization layers consist of the last 3 basic blocks (42 layers).

For the second configuration, the base layers consist of 1 residual block (6 layers) , 2 basic blocks (24 layers) and another 12 layers from the 3rd basic block. The personalisation layers consist of the remaining 6 layers of 3rd basic block and 5 basic blocks. In Figure 4, we showed the 42 layers of basic blocks (representing base layers for FedPer) structure of a ResNet18.

In Section 5.6, we explore the FedPer performance with a varying number of base layers and personalization layers. In particular, using ResNet18 as the baseline architecture, we compare the above two configurations.

Figure 4

Sample 42 Base Layers for FedPer



```
list(w_glob.keys())[:base_layers]

['backbone.conv1.weight',
'backbone.bn1.weight',
'backbone.bn1.bias',
'backbone.bn1.running_mean',
'backbone.bn1.running_var',
'backbone.bn1.num_batches_tracked',
'backbone.layer1.0.conv1.weight',
'backbone.layer1.0.bn1.weight',
'backbone.layer1.0.bn1.bias',
'backbone.layer1.0.bn1.running_mean',
'backbone.layer1.0.bn1.running_var',
'backbone.layer1.0.bn1.num_batches_tracked',
'backbone.layer1.0.conv2.weight',
'backbone.layer1.0.bn2.weight',
'backbone.layer1.0.bn2.bias',
'backbone.layer1.0.bn2.running_mean',
'backbone.layer1.0.bn2.running_var',
'backbone.layer1.0.bn2.num_batches_tracked',
'backbone.layer1.1.conv1.weight',
'backbone.layer1.1.bn1.weight',
'backbone.layer1.1.bn1.bias',
'backbone.layer1.1.bn1.running_mean',
'backbone.layer1.1.bn1.running_var',
'backbone.layer1.1.bn1.num_batches_tracked',
'backbone.layer1.1.conv2.weight',
'backbone.layer1.1.bn2.weight',
'backbone.layer1.1.bn2.bias',
'backbone.layer1.1.bn2.running_mean',
'backbone.layer1.1.bn2.running_var',
'backbone.layer1.1.bn2.num_batches_tracked',
'backbone.layer2.0.conv1.weight',
'backbone.layer2.0.bn1.weight',
'backbone.layer2.0.bn1.bias',
'backbone.layer2.0.bn1.running_mean',
'backbone.layer2.0.bn1.running_var',
'backbone.layer2.0.bn1.num_batches_tracked',
'backbone.layer2.0.conv2.weight',
'backbone.layer2.0.bn2.weight',
'backbone.layer2.0.bn2.bias',
'backbone.layer2.0.bn2.running_mean',
'backbone.layer2.0.bn2.running_var',
'backbone.layer2.0.bn2.num_batches_tracked']
```

FedProx: While updates are done on FedAvg and FedPer during the aggregation step, FedProx is a modification of the client local training process. It is a novel approach used to handle convergence improvements with the help of a proximal term to enhance local client performances in spite of highly diverse data. This proximal term is chosen based on

the dataset characteristics. During the client local updates, for every round of training we minimize the loss function F using a proximal term μ as follows [13]:

$$F = F_k(w) + \frac{\mu}{2} \|w - w^t\|^2 \quad (2)$$

where F_k is the old loss, and μ is the proximal term which varies according to the data. w is the client local data weight parameters and w^t is the global model parameters at time t . So by updating the loss of each round with 2-norm between global model and local model, and a proximal term to stabilize the model, we implement the slight modification to FedAvg method. The intuition behind this algorithm is that the proximal term will help in keeping the client local updates as close as they can to the initial global model and this will allow the algorithm to handle the issue of data heterogeneity among the clients. By varying the proximal term and with a number of epochs, clients show convergence in their performance with greater accuracies [13]. In our experiments, we use four different proximal values for this method.

Chapter 5

Empirical Results

In this chapter, we give a brief description of all the datasets that are used, the data preprocessing steps, the experimental scenarios, performance measures, and the experimental design and setups. Finally the experimental results are described and discussed in detail.

5.1 Datasets Description

Three datasets, namely Fashion MNIST [10], Omniglot [11], and CIFAR-100 [9] are used in our empirical study of the federated few-shot learning problem.

- 1) Fashion MNIST [10]: It is a dataset of Zalando's article images which consists of 60,000 images in training set and 10,000 images in testing set. All these images belong to 10 different classes and all are different types of clothes such as trousers, shirts etc. Every image is a 28 x 28 grayscale image.
- 2) Omniglot [11]: It is a dataset of 1623 hand-written characters from different languages written by 20 different persons, that is $1623 \times 20 = 32,460$ data points. It consists of characters from 50 different alphabet series. Every omniglot character image is 105*105 pixel size grayscale image. The training set consists of 19,280 data points and the test set consists of 13,180 data points.
- 3) CIFAR-100 [9]: It is a subset of 80 million tiny images dataset [9]. It consists of 60,000 images divided between 100 different classes. Each image is 32 x 32 pixels and is colored. The training set has 50,000 images of 500 classes and the test set has 10,000 images of 100 classes.

5.2 Dataset Preprocessing

Experiments are performed on Fashion MNIST, Omniglot, and CIFAR-100 to study federated few-shot learning. The n-way, n-shot and n-query parameters are manually selected by the user during training and testing. For our experiments, we use three few-shot learning task configurations, namely: 3-way: 5-shot: 10-query, 5-way: 5-shot: 10-query and 5-way: 5-shot: 5-query.

The three datasets are directly imported from the Pytorch Python package. We then download them, transform them into tensor data by normalizing, and then load them into different sets of dataloaders for the clients and server to utilize them more easily for training and testing. We used pytorch because it already has every dataset pre-divided into training and test sets and is easy for transforming and composing according to our needs. Since we are considering different non i.i.d forms in a simulated federated learning scenario we chose this simplest way for data division among clients and the server.

5.3 Experimental Setups

We explore the working and performance on the three datasets for the two client-server scenarios: (a) 2-client and 1 server, and (b) 3-client and 1 server.

We analyze the performances of different approaches on these datasets in two ways: individually and collectively. For the first way, we perform meta-learning and federated learning among clients and the server for each dataset individually and change the shots and query sets of each task. Every dataset is divided into five parts namely, one large part for server base training, three parts for clients individual training and finally one part for final server testing. These five parts are the same for every round but the few-shot tasks for

each round are chosen randomly using the data loader from these parts. The data distribution in this scenario depends on the number of clients and are described below.

- 1) Fashion MNIST: The server base training set and three client local data parts are taken from the training set and the server final testing would be on the shots taken from the test set only. Server base training has 30,000 image data points. The remaining 30,000 data points are divided equally among the three clients (i.e. 10,000 images for each client). During the training rounds, different n-way k-shot q-query sets will be chosen randomly from these individual data parts.
- 2) Omniglot: This is the smallest dataset with only 1623 hand-written characters from different languages written by 20 different people. Since each client chooses shots randomly, the data loader needs sufficiently more data but the dataset here is comparatively smaller than others. Hence, we only perform experiments on the 2 clients and 1 server scenario.
- 3) CIFAR-100: Similar to Fashion MNIST, we divide the training set into 4 parts. Server base training gets 20,000 images and the remaining 30,000 data points are equally divided among the three clients. The test set is given to the server testing.

5.4 Performance Measures

Like all other typical image classification problems, we use accuracy to measure the performance of different models. Accuracy gives the percentage of correct predictions out of all the total predictions. Since the datasets that we are working on are balanced and this is clearly a problem of multi-class classification, this performance metric would be

very useful. This metric provides useful information about how often the classifier can predict correctly.

5.5 Experimental Designs and Implementation

Since we do not assume that all devices involved in the knowledge sharing of model parameters have sufficient computational resources, we only train every model for 20 rounds and the number of epochs for each client or server is 1. Number of episodes in pre-training is 400. The optimization technique we use in the ResNet18 [35] model is Stochastic Gradient Descent (SGD) [36]. It is a widely used optimizer for convolutional deep neural networks and it is best in finding optimal solutions iteratively and to decrease the loss. The step size of weight updation or the learning rate hyperparameter for this algorithm that we have chosen is 0.1 for all experiments. The platform used for the experiments is Google Colaboratory Pro+ which consists of a GPU (NVIDIA PT100), 52 GB RAM for faster runtime and efficient computation.

5.6 Experimental Results and Discussions

5.6.1 Performance Comparison of 2 Layer Ratios in FedPer Using Fashion-MNIST

The FedPer algorithm allows the client to concentrate more on learning and personalizing on its own data by dividing the client model layers into two parts: base and personalization layers (see Section 4.4). Maintaining an appropriate ratio of base layers is an essential criteria here. In this experiment, we compare 2 configurations: (1) 78 base layers and 42 personalized layers and (2) 42 base layers and 78 personalization layers using a 2-client and one server experiment scenario.

Table 1*Base+Personalization Layers Effect on Fashion-MNIST (2 Clients)*

Base + Personalization layers	N-way classes	K-shot	Q-Query	Server training accuracy on model M	C1 accuracy on S1-Q1	C2 accuracy on S2-Q2	Server testing accuracy on Aggregated model M'
42+78	3	5	10	69.950	79.0	75.499	86.66
78+42				67.801	71.166	75.0	88.33
42+78	5	5	10	61.355	68.2	69.8	84.0
78+42				62.635	69.2	68.6	85.1
42+78	5	5	5	57.89	67.6	67.2	78.8
78+42				56.270	66.8	67.4	77.8

From Table 1, we observe that their prediction performance is comparable for the three different few-shot learning task configurations on the server global model and the client local models. When we are performing experiments using FedPer for our proposed solution, we use the model with the first configuration (78 base and 42 personalized layers) which emphasizes less on the importance of local data. In the next section, we see the impact of FedProx parameter changes on the same dataset.

5.6.2 Effect of FedProx Proximal Term μ on Prediction Performance Using Fashion-MNIST

The proximal term μ used in FedProx algorithm helps stabilize the machine learning model when there is a need for each client device to train for a number of epochs. It helps to optimize the performance of each client using its local data (see Section 4.4).

Figure 5

Prediction Performance as μ Varies on Fashion-MNIST (2 Clients)

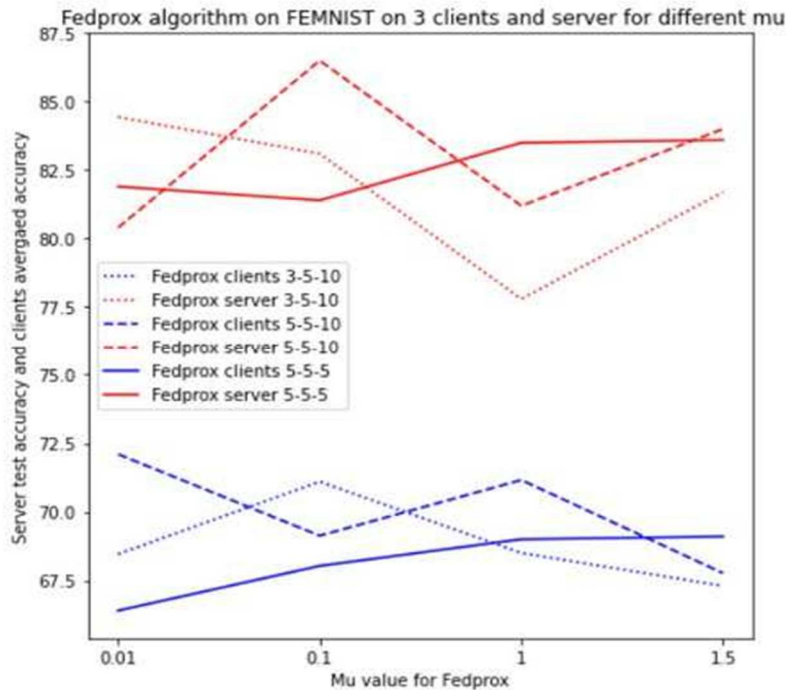


Figure 5 shows the average of all the three clients and server prediction performance on three few-shot learning task configurations on the Fashion-MNIST dataset over all 20

rounds as μ takes the values 0.01, 0.1, 1 and 1.5. One general observation (also for all our subsequence experiments) is that the prediction performance for the server global model is consistently better than the client local models as more data (before or after the aggregation) are used if we use a single dataset. One observes that one cannot pick a particular μ value that works best across the different few-shot learning task configurations even if the configuration variation is not significant. We use $\mu=1$ in our experiments.

5.6.3 Performance Comparison of Proposed Solution Using Different Federated Learning on Each Dataset in Single Dataset Scenario

5.6.3.1 Fashion-MNIST. Table 2 shows the prediction performance for the server global models (before and after aggregation) and clients' local models for the three few-shot learning task configurations on the Fashion-MNIST dataset with the three different federated learning aggregation methods. In the table, Server Base Train M (column 5) is the training accuracy for models learned using the large base set on the server which happens episodically. Column 6, 7, 8, 9 are the average test accuracy for models for Client C1, C2, C3, and the aggregated model M' on server over 20 rounds.

The computation time is very minimum for 20 rounds. When it comes to a 2-client prediction, each experimental trial takes about 300 to 420 seconds. In the case of 3-clients, an experimental trial takes 600 to 696 seconds. Noticeably, the FedProx algorithm when computing for 3 clients takes the highest time of 696 sec and FedPer of 685 seconds.

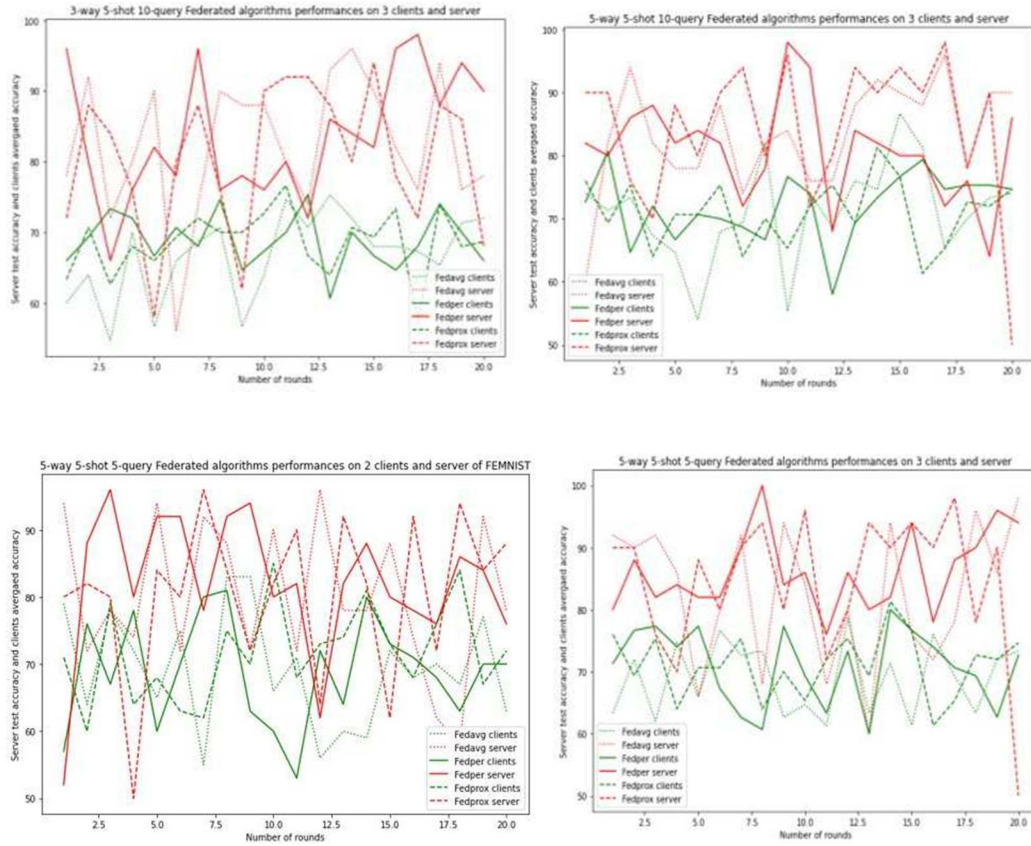
Table 2*Results on Fashion-MNIST Dataset*

No.of clients	Type of Algorithm	N-way classes	K-shot	Q-Query	Server training accuracy on model M	C1 accu on S1-Q1	C2 accu on S2-Q2	C3 accu on S3-Q3	Server testing accuracy on Aggregated model M'
2	FedAvg	3	5	10	61.195	64.90	71.50	-	80.10
3	FedAvg				60.13	67.0	66.10	67.3	82.6
2	FedPer				60.725	76.4	68.0	-	87.3
3	FedPer				61.68	71.7	65.2	69.8	83.7
2	FedProx				61.225	70.00	67.10	-	79.70
3	FedProx				59.91	65.40	72.40	68.20	80.60
2	FedAvg	5	5	10	61.08	75.20	67.80	-	82.90
3	FedAvg				61.52	70.70	70.4	72.4	83.3
2	FedPer				61.335	72.40	70.2	-	79.4

No.of clients	Type of Algorithm	N-way classes	K-shot	Q-Query	Server training accuracy on model M	C1 accu on S1-Q1	C2 accu on S2-Q2	C3 accu on S3-Q3	Server testing accuracy on Aggregated model M'
3	FedPer				61.3	70.3	75.4	70.8	89.0
2	FedProx				61.22	65.40	67.60	-	80.80
3	FedProx				61.385	66.0	70.4	71.0	86.5
2	FedAvg	5	5	5	62.0	69.60	68.70	-	80.30
3	FedAvg				60.585	68.60	67.40	70.70	82.60
2	FedPer				59.805	67.70	69.90	-	81.90
3	FedPer				60.76	69.40	71.90	71.90	86.10
2	FedProx				60.58	71.00	72.30	-	80.50
3	FedProx				61.75	67.80	68.40	69.50	80.30

Figure 6

Fashion MNIST Single-Data Results



Note. Top Left: 3-5-10; Top Right: 5-5-10; Bottom Left: 5-5-5 (3 clients); Bottom Right: 5-5-5 (2 clients)

From Table 2, we observe that FedPer aggregated global models consistently perform the best for the three few-shot learning task configurations. Moreover, recall that the server has a large base set and their training accuracies were only around 60%. After federated learning, we observe a significant improvement in the performance of the server after aggregations on entirely previously unseen test data compared to the clients' models. None

of the three federated learning methods outperforms the others as their aggregated global models do not consistently help improve clients' predictive performance. However, we did observe that performance for all client (fine-tuned) local models improved from performance of the global model sent to the client (before fine-tuning).

In Figure 6, we again see that the server global model testing performance for the few-shot learning task is always greater than the client's local model. However, there is no clear winner on which aggregation method is best for this dataset according to Figure 6. In fact, there is no consistent improvement (or convergence) in prediction performance for the few-shot learning task as more rounds (i.e., more meta-learning and federated learning) are iterated.

5.6.3.2 CIFAR-100. From Table 3, we observe that few-shot learning tasks constructed from the CIFAR-100 dataset are very challenging tasks with average accuracy between 24% and 48%. Moreover, none of the federated learning approaches is favorable for few-shot learning tasks constructed from this dataset. While the aggregated models performed better than the initial models learned via meta-learning, the client local models did not perform better than the initial models sent to the clients. In Figure 7 we observe that unlike the Fashion MNIST few-shot learning tasks, the aggregated global models do not performed significantly better than the client fine-tuned models for the three few-shot learning task configurations on the CIFAR-100 dataset with the three different federated learning aggregation methods. The computation time is very minimum for 20 rounds. Noticeably, the FedProx algorithm when computing for 3 clients takes the highest time of 571 sec and FedPer of 609 seconds both less than Fashion-MNIST.

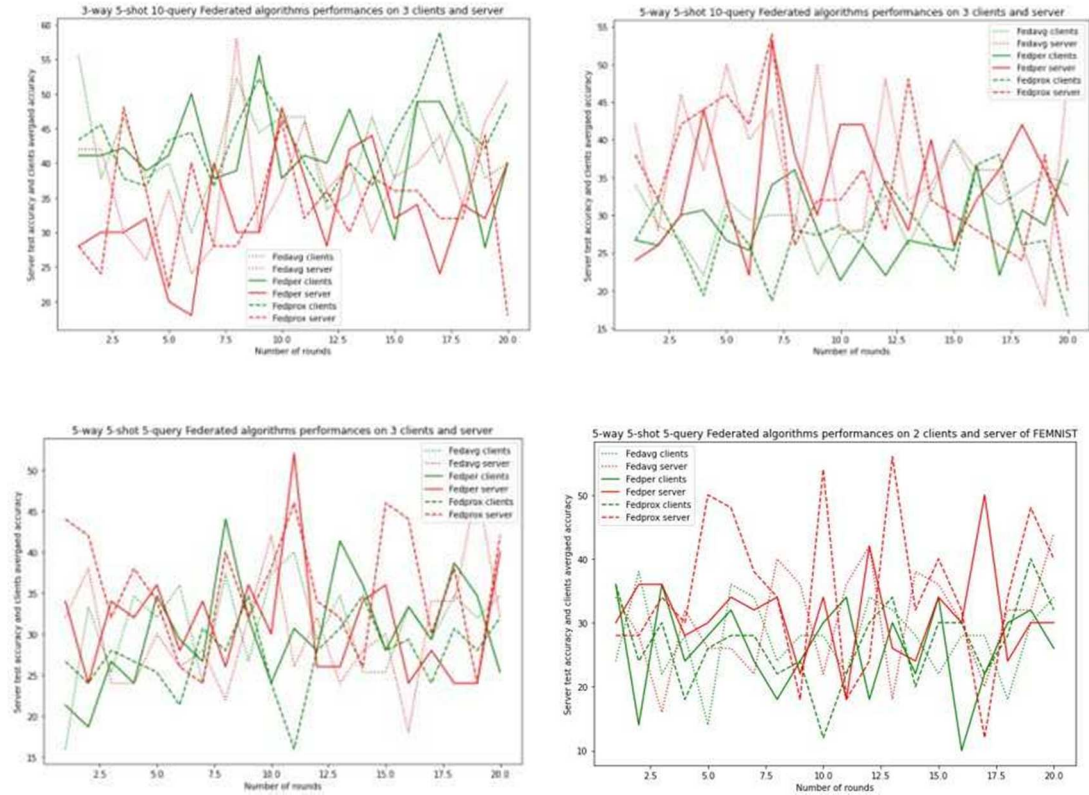
Table 3*Results on CIFAR-100 Dataset*

No. of clients	Type of Algorithm	N-way classes	K-shot	Q-Query	Server training accu on model M	C1 accu on S1-Q1	C2 accu on S2-Q2	C3 accu on S3-Q3	Server testing accu on Aggregated model M'
2	FedAvg	3	5	10	40.258	41.33	38.66	-	31.0
3	FedAvg				40.775	39.83	44.66	42.33	37.90
2	FedPer				40.758	43.33	40.33	-	29.3
3	FedPer				41.858	47.33	38.16	38.83	32.7
2	FedProx				41.575	44.33	42.83		30.30
3	FedProx				41.75	45.33	42.66	43.00	33.40
2	FedAvg	5	5	10	29.240	32.0	28.8	-	30.4
3	FedAvg				29.685	32.50	29.5	29.4	37.20
2	FedPer				29.02	30.6	27.3	-	39.80
3	FedPer				28.29	27.7	29.8	27.4	34.2

No. of clients	Type of Algorithm	N-way classes	K-shot	Q-Query	Server training accu on model M	C1 accu on S1-Q1	C2 accu on S2-Q2	C3 accu on S3-Q3	Server testing accu on Aggregated model M'
2	FedProx				29.67	30.0	30.50	-	32.80
3	FedProx				28.095	26.00	30.30	25.80	34.90
2	FedAvg	5	5	5	28.49	30.80	24.40	-	30.50
3	FedAvg				27.59	31.40	31.20	30.60	30.20
2	FedPer				27.23	25.20	27.20	-	31.20
3	FedPer				26.35	26.2	31.8	33.0	31.4
2	FedProx				28.87	25.80	27.80	-	34.90
3	FedProx				26.84	26.20	24.80	31.60	35.60

Figure 7

CIFAR-100 Single-Data Results



Note. Top Left: 3-5-10; Top Right: 5-5-10; Bottom Left: 5-5-5 (3 clients); Bottom Right: 5-5-5 (2 clients)

5.6.3.3 Omniglot. The omniglot dataset consists of much fewer data points compared to the other two datasets, hence we only considered the case of 2-clients. From Table 4, one observes that FedPer and FedProx resulted in significant improvement in the global model performance. All three federated learning algorithms with meta-learning resulted in improvements in the client’s local model prediction performance. From Table

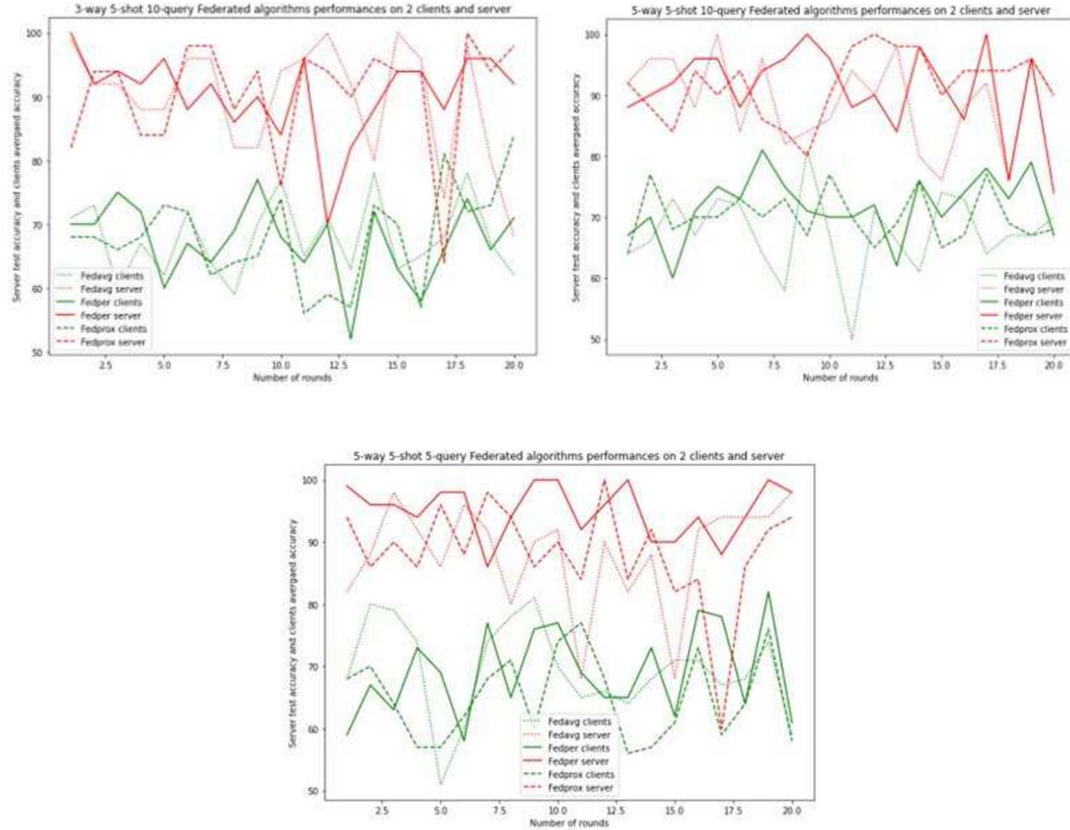
3 and Figure 8, we observed that the aggregated global model performances are better than the client local model prediction performance. Similar to Fashion MNIST, there is no consistent improvement (or convergence) in prediction performance for the few-shot learning task as more rounds (i.e., more meta-learning and federated learning) are iterated.

Table 4*Results on Omniglot Dataset*

No.of clients	Type of Algorithm	N-way classes	K-shot	Q-Query	Server train accu on model M	C1 Accu on S1-Q1	C2 Accu on S2-Q2	Server testing accuracy on Aggregated model M'
2	FedAvg	3	5	10	57	64.7	70.8	89.7
2	FedPer				55.93	70.40	64.40	90.50
2	FedProx				56.445	67.400	68.80	90.60
2	FedAvg	5	5	10	57.315	67.80	67.00	89.20
2	FedPer				58.605	69.60	73.80	91.00
2	FedProx				58.67	71.6	68.6	91.7
2	FedAvg	5	5	5	57.715	70.80	68.00	88.20
2	FedPer				58.185	69.90	68.30	95.20
2	FedProx				57	68.2	61.8	88.3

Figure 8

Omniglot Single-Data Results



Note. Top Left: 3-5-10; Top Right: 5-5-10; Bottom: 5-5-5

5.6.4 Performance Comparison of Proposed Solution Using Different Federated Learning in Multiple Datasets Scenario

We consider the case when multiple datasets are used across the server and client devices to explore the prediction performance of the global model under the heterogeneous data scenario using meta-learning to solve the few-shot learning tasks. Here, we consider the

server training on CIFAR-100 and Omniglot. Testing is performed on the global model and client local models that are used to predict on a single dataset for few-shot learning.

Table 5*Results on Multiple-Data Scenario of All Three Datasets*

No.of clients	Type of Algo	N-way classes	K-shot	Q-query	Server M Train Accuracy CIFAR, Omniglot	C1 S1-Q1 Cifar Accuracy	C2 S2-Q2 Omniglot Accuracy	C3 S3-Q3 Fash-Mnist Accuracy	Server Test M' Accu F-MNIST Accuracy	Server Test M' Accuracy CIFAR	Server Test M' Accuracy Omniglot
2	Fed Avg	3	5	10	63.933	41.33	90.00	-	66.50	41.16	79.33
3	Fed Avg				64.65	40.33	90.66	60.33	74.00	44.83	83.00
2	Fed Per				64.191	46.33	92.00	-	64.16	48.50	71.50
3	Fed Per				61.975	44.83	87.166	55.66	69.166	45.66	81.00

No.of clients	Type of Algo	N-way classes	K-shot	Q-query	Server M Train Accuracy CIFAR, Omniglot	C1 S1-Q1 Cifar Accuracy	C2 S2-Q2 Omniglot Accuracy	C3 S3-Q3 Fash-Mnist Accuracy	Server Test M' Accuracy F-MNIST Accuracy	Server Test M' Accuracy CIFAR	Server Test M' Accuracy Omniglot
2	Fed Prox				64.2083	48.83	91.00	-	79.00	45.66	87.833
3	Fed Prox				65.1416	44.833	88.50	64.33	73.66	47.83	86.53
2	Fed Avg	5	5	10	60.62	32.30	91.40	-	59.90	30.30	87.50
3	Fed Avg				58.545	31.00	87.50	56.30	62.60	29.60	74.90

No.of clients	Type of Algo	N-way classes	K-shot	Q-query	Server M Train Accuracy CIFAR, Omniglot	C1 S1-Q1 Cifar Accuracy	C2 S2-Q2 Omniglot Accuracy	C3 S3-Q3 Fash-Mnist Accuracy	Server Test M' Accu F-MNIST Accuracy	Server Test M' Accuracy CIFAR	Server Test M' Accuracy Omniglot
2	Fed Per				57.365	32.40	88.80	-	53.30	27.70	87.10
3	Fed Per				59.925	31.70	88.90	51.00	64.60	34.30	67.90
2	Fed Prox				58.16	27.90	89.60	-	49.70	28.30	85.5
3	Fed Prox				56.56	30.00	86.5	50.10	60.60	33.10	74.80
2	Fed Avg	5	5	5	53.02	26.285	85.904	-	60.60	33.40	76.20

No.of clients	Type of Algo	N-way classes	K-shot	Q-query	Server M Train Accuracy CIFAR, Omniglot	C1 S1-Q1 Cifar Accuracy	C2 S2-Q2 Omniglot Accuracy	C3 S3-Q3 Fash-Mnist Accuracy	Server Test M' Accu F-MNIST Accuracy	Server Test M' Accuracy CIFAR	Server Test M' Accuracy Omniglot
3	Fed Avg				56.6	27.60	87.0	49.60	67.4	32.80	69.60
2	Fed Per				52.19	27.60	87.00	-	67.4	32.80	84.20
3	Fed Per				53.05	33.20	88.00	49.40	62.800	33.80	71.60
2	Fed Prox				53.11	31.80	84.20	-	47.80	30.00	80.60
3	Fed Prox				53.18	29.80	86.20	49.20	66.80	32.30	77.200

During client fine-tuning, each client will be given a single dataset. *We investigate how the global model reacts to few-shot learning tasks on an unseen dataset, but only provided relevant information to the client model fine-tuning.* The results for this scenario are shown in Table 5.

Some observations from Table 5 are:

Broadly comparing single-dataset scenarios and multiple-dataset scenarios with respect to server testing accuracies, one can make the following observations.

- 1) Fashion-MNIST test accuracies for single-dataset scenarios are in the range of 80% to 86% where in case of multiple-dataset scenarios it is only 60% to 70% in all few-shot learning task configurations.
- 2) CIFAR-100 under multiple-dataset scenarios has accuracies in the range of 27% to 33% in case of 5-5-5 and 5-5-10 few-shot learning task configuration and 40% to 48% in case of 3-5-10 few-shot learning task configuration, whereas in single-dataset case it is just 30% to 35% in all three few-shot learning task configurations.
- 3) Omniglot in single-dataset scenarios is more accurate in the range of 88% to 90% whereas in multiple-dataset scenarios its accuracy is only between 71% and 87%.

From the above three observations, we can say that Fashion-MNIST and Omniglot have decreased their performance in multiple data scenarios, whereas CIFAR-100 has shown the same or a slightly increased performance compared to single data scenarios.

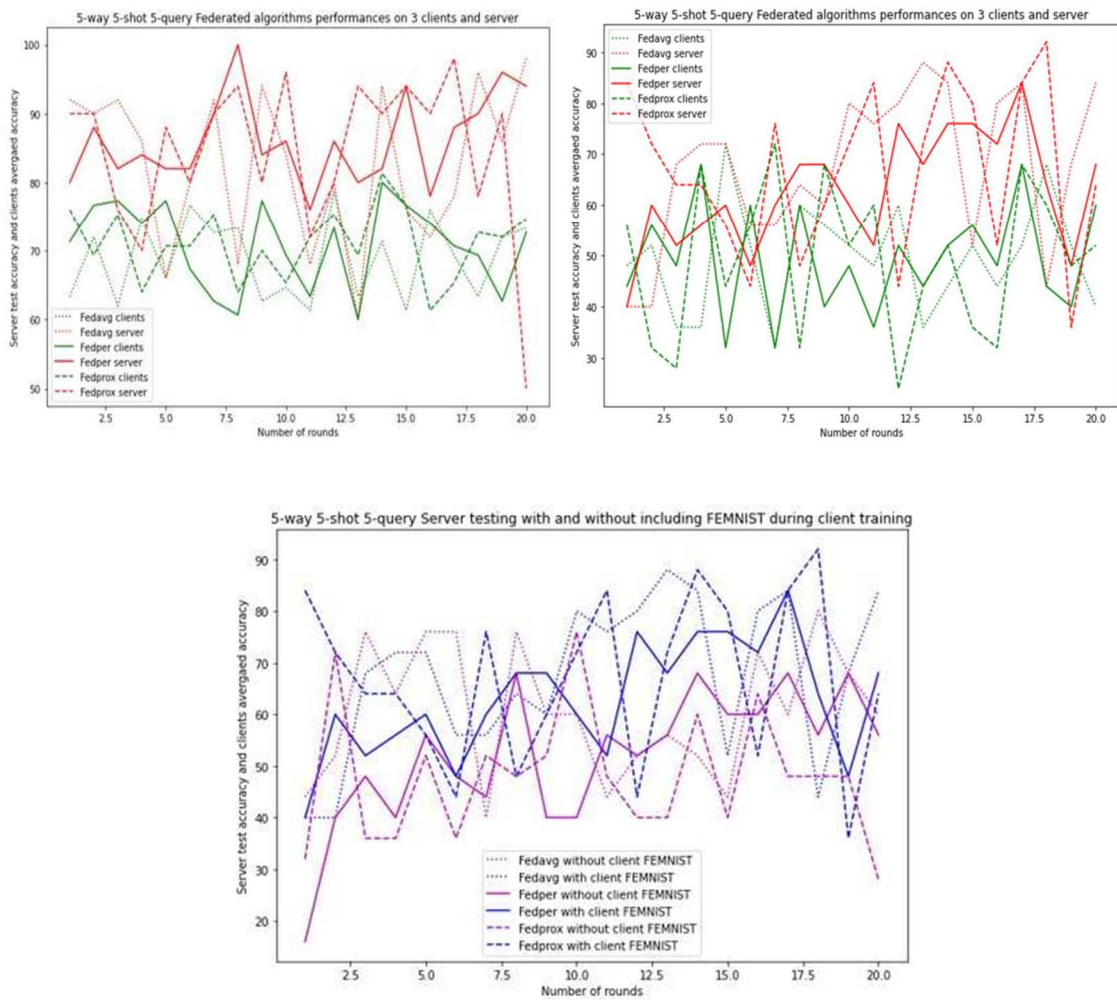
Next, we will see how few-shot learning for each dataset varied from single-dataset to multiple-dataset scenario over the 20 federated rounds. We only use 5-5-5 task configuration for algorithm-wise comparisons in both the scenarios. For the next three

figures, on the left side we see the case of single-dataset (from Figure 5, 6, 7) and on the right side the case of multiple-dataset.

5.6.4.1 Fashion-MNIST.

Figure 9

Results of Fashion-MNIST on Single-Data and Multiple-Data Scenarios



Note. Top-Left: Single-Dataset Scenario; Top-Right: Multiple-Dataset Scenarios Results;

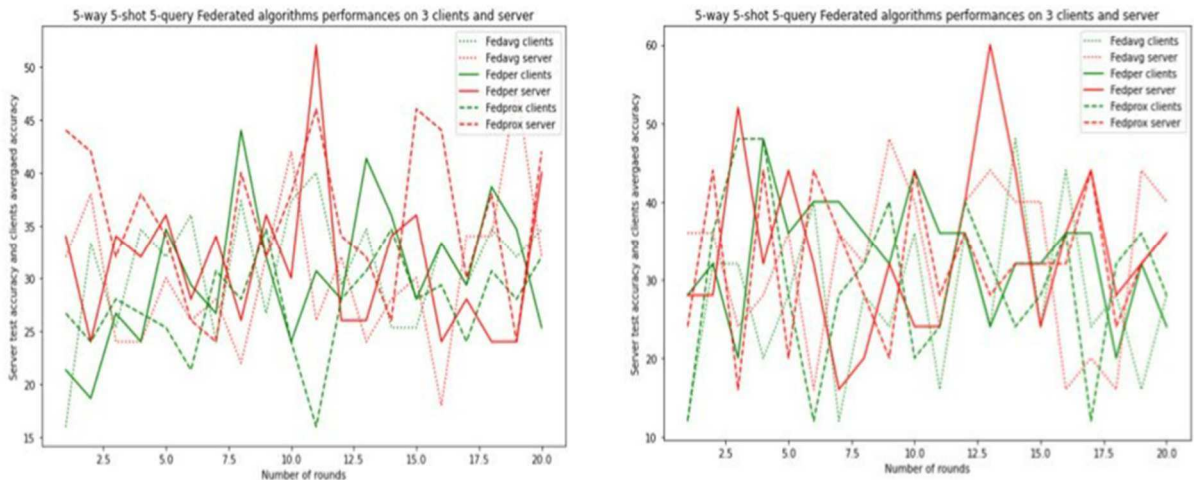
Bottom: With and Without Client Training On Fashion-MNIST

Firstly, Figure 9 (top right side) shows the server performance for 20 rounds with and without client training on few-shot learning tasks using Fashion-MNIST. When the client has not been trained on Fashion-MNIST, server performance is not better. When the client has been trained on Fashion-MNIST, the server model trained using FedProx is the best performer over all three aggregation algorithms. Figure 9 (bottom) shows the client and server testing performances. In some rounds, server testing accuracy is greater than client, whereas in some rounds it is not.

5.6.4.2 CIFAR-100. In Figure 10, we compare the individual client and server performances on all 20 rounds in both the single-dataset and multiple-dataset scenarios of CIFAR-100 dataset.

Figure 10

Comparison of CIFAR-100 on Single-Data and Multiple Data Scenarios



Note. Top-Left: Single-Data Results; Top-Right: Multiple-Data Results

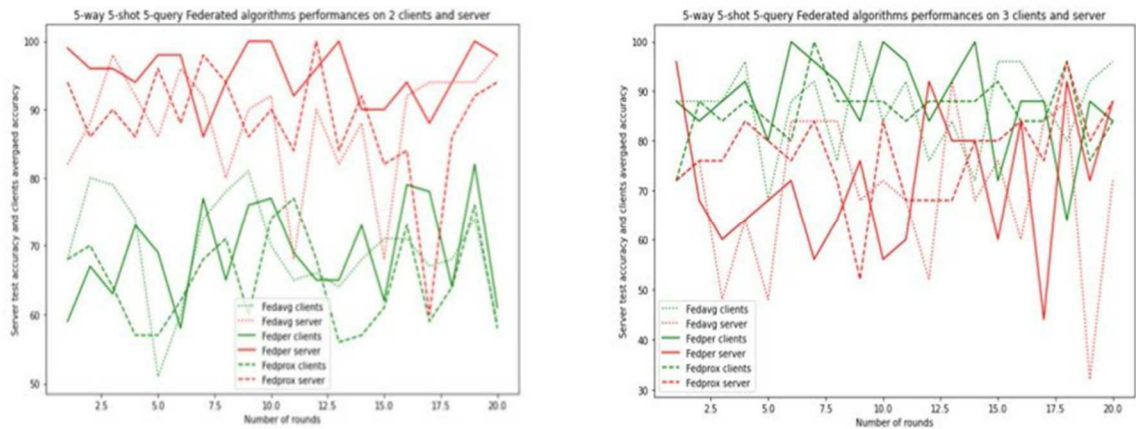
There is no significant difference between client and server performances in both scenarios.

5.6.4.2 Omniglot. In Figure 11, we see that the individual client accuracies are higher compared to server testing accuracies in case of multiple-datasets which is not the case in single-data scenario. In other words, the Omniglot dataset has a rapid decrease in its server performance when it comes to multiple-datasets.

Also, there is no particular algorithm that performs constantly better in both the cases for this dataset.

Figure 11

Comparison of Omniglot on Single-Dataset and Multiple-Dataset Scenarios



Note. Top-Left: Single-Data Results; Top-Right: Multiple-Data Results

Chapter 6

Conclusions and Future Work

In this thesis, we described our study of combining federated learning and meta-learning to handle independent few-shot learning tasks on multiple devices and the server. In particular, we proposed utilizing the prototypical networks to perform meta-learning on all devices to learn multiple independent few-shot learning models and to combine the models in a centralized architecture using federated learning which can be reused by the clients subsequently. We performed extensive experiments to (1) compare three different federated learning approaches, namely Federated Averaging (FedAvg), Federated Proximal (FedProx), and Federated Personalization (FedPer) on our proposed framework, and (2) explore the effect of data heterogeneity on the few-shot learning performance. Our empirical results show that our proposed approach is feasible and is able to improve the edge devices' individual prediction performance and improve significantly on the global model (on the server) using any of the federated learning approaches when the few-shot learning tasks are from the same datasets. However, the data heterogeneity problem still affects the prediction performance of our proposed solution no matter which federated learning approach we used.

In this thesis, we assume that the server and clients may have data from the same classes in the few-shot learning tasks. In other words, the datasets are partitioned for the server and clients in our experiments, but the partitioning did not take into account the fact that the server and clients should have a non-overlapping set of classes. The most important

part of future work is to perform experiments to study meta-few-shot learning in federated learning scenarios by dividing the classes for base-training set and support/query sets such that the server and clients will not have data from the same class.. This will provide additional experimental scenarios to study the data heterogeneity issue and additional meta-learning algorithms can be investigated. In addition, other federated learning algorithms can be compared using more clients. Exploring the federated meta-few-shot learning with decentralized architectures can be another interesting topic of research. Unlike inductive meta-learning algorithms, implementing this concept with transductive meta-learning algorithms can significantly improve the performance of few-shot learning and implementing this in a federated scenario would be another possible future work.

References

- [1] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a Few Examples," *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–34, 2020.
- [2] D.C. Nguyen, P.N Pathirana & J.Li," Federated Learning for Internet of Things: A Comprehensive Survey" *IEEE Communications and Surveys*, vol. 23, issue.3, pp. 1622-1658, IEEE, 2021.
- [3] W. Y. B. Lim et al., "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," in *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031-2063, 2020.
- [4] T. M. Hospedales, A. Antoniou, P. Micaelli, and A. J. Storkey, "Meta-Learning in Neural Networks: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 09, pp. 5149–5169, 2021.
- [5] M. A. Jamal and G. -J. Qi, "Task Agnostic Meta-Learning for Few-Shot Learning," in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 11711-11719.
- [6] Y. Chen, Z. Liu, H. Xu, T. Darrell and X. Wang, "Meta-Baseline: Exploring Simple Meta-Learning for Few-Shot Learning," in 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 9042-9051.
- [7] J. Snell, K. Swersky, and R. Zemel " Prototypical Network for Few-Shot Learning," in Proceedings of 31st International Conference on Neural Information Processing Systems (NIPS' 2017), Red Hook, NY, USA, 2017, pp.4080-4090.
- [8] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *ICML '17: Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 1126–1135, 2017.
- [9] *Benchmarks CIFAR-100*. (n.d.). [Database]. CIFAR-100; benchmarks. ai. <https://benchmarks.ai/cifar-100>.
- [10] *Benchmark dashboard*. (n.d.). Retrieved May 15, 2022, from <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/>
- [11] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.
- [12] B. H. McMahan, E. Moore, D. Ramage, S. Hampson, & B. A. Arcas,

- “Communication-Efficient Learning of Deep Networks from Decentralized Data”. Presented at the Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017. JMLR: W&CP, 54.
- [13] T. Li, A. K. Sahu, M. Zaheer, M. Sunjabi, A. Talwalkar, & V. Smith, “Federated Optimization In Heterogeneous Networks” presented at the Proceedings of the 3 rd MISys Conference, Austin, TX, USA, 2020.
- [14] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, “Federated Learning with Personalization Layers,” *arXiv:1912.00818[cs, stat]*, 2019.
- [15] H. Yu, I. Mineyev, L. R. Varshney, and J. A. Evans, “Learning from One and Only One Shot,” *arXiv:2201.08815[cs]*, 2022.
- [16] F. Pourpanah et al., "A Review of Generalized Zero-Shot Learning Methods," in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022, doi: 10.1109/TPAMI.2022.3191696.
- [17] W. Chen, Y. C. Liu, Z. Kira, & J. B. Huang, “A Closer Look at Few-shot Classification” presented at the International Conference on Learning Representations (ICLR), 2019.
- [18] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, “Generalizing from a Few Examples,” *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–34, 2020.
- [19] C. Fan and J. Huang, "Federated Few-Shot Learning with Adversarial Learning," in 19th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt), 2021, pp. 1-8.
- [20] K. Li, Y. Zhang, K. Li and Y. Fu, "Adversarial Feature Hallucination Networks for Few-Shot Learning," in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 13467-13476.
- [21] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, & Y. Khazaeni, “Federated Learning with Matched Averaging”, presented at the International Conference on Learning Representations (ICLR), 2020.
- [22] S. Ek, F. Portet, P. Lalanda and G. Vega, "A Federated Learning Aggregation Algorithm for Pervasive Computing: Evaluation and Comparison," 2021 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2021, pp. 1-10.
- [23] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, “Exploiting Shared Representations for Personalized Federated Learning,” in *Proceedings of the 38 th International Conference on Machine Learning*, 2021, vol. 139, pp. 2089–2099.

- [24] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, “Federated Meta-Learning with Fast Convergence and Efficient Communication,” *arXiv:1802.07876[cs]*, 2019.
- [25] A. Fallah, A. Mokhtari, & A. Ozdaglar, “Personalized Federated Learning: A Meta-Learning Approach” presented at the 34th Conference on Neural Information Processing Systems NeurIPS, Vancouver, Canada, 2020.
- [26] “LEAF: A Benchmark for Federated Settings,” *GitHub*, Sep. 28, 2021.
<https://github.com/TalwalkarLab/leaf>
- [27] J. Li, M. Khodak, S. Caldas, & A. Talwalkar, “Differentially Private Meta-Learning”, presented at the International Conference on Learning Representations (ICLR), 2020.
- [28] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, , J. B. Tenenbaum, H. Larochelle, & R. S. Zemel, Meta-Learning for Semi-Supervised Few-Shot Classification. In *Proceedings of 6th International Conference on Learning Representations (ICLR)*, 2018.
- [29] Y. Hu, V. Gripon, and S. Pateux, “Leveraging the Feature Distribution in Transfer-Based Few-Shot Learning,” *Lecture Notes in Computer Science*, pp. 487–499, 2021.
- [30] Q. Li, Z. Wen, and B. He, “Practical Federated Gradient Boosting Decision Trees,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 4642–4649, Apr. 2020.
- [31] O. Vinyals, “Matching Networks for One Shot Learning”, presented at the Advances in Neural Information Processing Systems NIPS, 2016, vol 29.
- [32] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr and T. M. Hospedales, "Learning to Compare: Relation Network for Few-Shot Learning," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 1199-1208.
- [33] Y. Hu, S. Pateux, and V. Gripon, “Squeezing Backbone Feature Distributions to the Max for Efficient Few-Shot Learning,” *Algorithms*, vol. 15, no. 5, p. 147, 2022.
- [34] M. Boudiaf, “Information Maximization for Few-Shot Learning”, presented at the Advances in Neural Information Processing Systems, NeurIPS, Vancouver, Canada, 2020, vol 33, pp. 2445-57.
- [35] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.
- [36] D. Needell, N. Srebro, and R. Ward, “Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm,” *Mathematical Programming*, vol. 155, no. 1–2, pp. 549–573, 2015.

