FedNet2Net: Saving Communication and Computations in Federated Learning with Model Growing*

Amit Kumar Kundu and Joseph Jaja

University of Maryland, College Park, MD 20742, USA
{amit314,josephj}@umd.edu

Abstract. Federated learning (FL) is a recently developed area of machine learning, in which the private data of a large number of distributed clients is used to develop a global model under the coordination of a central server without explicitly exposing the data. The standard FL strategy has a number of significant bottlenecks including large communication requirements and high impact on the clients' resources. Several strategies have been described in the literature trying to address these issues. In this paper, a novel scheme based on the notion of "model growing" is proposed. Initially, the server deploys a small model of low complexity, which is trained to capture the data complexity during the initial set of rounds. When the performance of such a model saturates, the server switches to a larger model with the help of *function-preserving transformations*. The model complexity increases as more data is processed by the clients, and the overall process continues until the desired performance is achieved. Therefore, the most complex model is broadcast only at the final stage in our approach resulting in substantial reduction in communication cost and client computational requirements. The proposed approach is tested extensively on three standard benchmarks and is shown to achieve substantial reduction in communication and client computation while achieving comparable accuracy when compared to the current most effective strategies.

Keywords: Communication Efficiency · Federated Learning · Function Preserving Transformation.

1 Introduction

Federated learning (FL) is a new machine learning (ML) paradigm that enables the training of a model by utilizing private data distributed across many clients governed by a central server [16]. In contrast to the traditional ML, where all samples are stored in a single place, FL assumes that the data are generated and collected by many distributed, independent clients. Therefore, the overall data is expected to be heterogeneous and non-IID (Identically and Independently Distributed). In training, the server broadcasts the current global model

* Partially supported by a DOD contract to the University of Maryland Institute for Advanced Computer Studies.

to a set of randomly selected clients. Each selected client locally trains the received model with its private data and sends the updates to the server. The server aggregates the updates on the current model using federated averaging. This constitutes a single communication round. This procedure is repeated for many communication rounds, where in each round the randomly selected clients advance the training, until convergence is achieved. FL has been growing in importance especially with the emergence of edge computing and AI on the edge due to the massive deployment of IoT devices and advances in communication and networking systems [23, 30].

In the most general setting, the standard FL strategy has a number of significant bottlenecks that need to be addressed before it can be widely used in practice. These bottlenecks include data heterogeneity [5], unreliable and variable rate connectivity [16], uneven and relatively limited client resources [8], high communication requirement, and high impact on the clients' resources. Strategies to address these bottlenecks is suggested in the literature, see for example [11] but here we focus our attention on the two bottlenecks of communication requirements and the limited client resources.

Three approaches have been suggested to reduce the communication requirement and achieve comparable performance to the standard FL. The first, as in [6, 18, 28], relies on quantization methods; the second relies on the sparsification of the model update as in [12]; and the third approach broadcasts smaller networks to improve communication efficiency [13, 27].

Another constraint is the relatively limited resources available at the clients. In general, whenever a state-of-the-art model is required to capture the global heterogeneous data complexity, FL induces a significant computational overhead on the clients. Therefore, we should aim at reducing the computational requirements on the clients. In [3], federated dropout is introduced in which random sub-networks of the entire model are broadcast thereby reducing communication bandwidth and computational resources.

The main contributions of this paper are:

- A novel strategy called FedNet2Net is introduced in which we start with a small initial model, and gradually enlarge the model to capture the increasing complexity of the data processed by the clients and improve accuracy.
- Function preserving transformations are used to switch from one model to the next once the performance of the current model saturates. Our switching is efficient and ensures a continuous improvement in accuracy as long as the inherent complexity of the data increases.
- FedNet2Net is shown through extensive experiments to result in large savings in the amounts of computation and communication compared to several of the best known strategies.
- FedNet2Net can be used to adaptively terminate at the smallest possible model that achieves the desired accuracy.

The rest of the paper is organized as follows. Section 2 provides an overview of techniques related to reducing the communication and client resources, while Section 3 describes our approach in details. Section 4 introduces the benchmarks

used for evaluation and describes the details of our model implementations. We present and discuss the results in Section 5 and we conclude in Section 6.

2 Related Work

Since the introduction of federated averaging [19], reducing communication bandwidth and local computation has received a great deal of interest in the community. Most of the work tried to improve communication efficiency, which can be broadly categorized into model compression based techniques [6, 15, 18, 20, 21, 28, 31], update sparsification based techniques [12] or broadcasting smaller networks [10, 13, 27]. Model compression techniques involve mainly quantization [6, 18, 28], compressed sensing [15], low rank approximation [20] and tensor decomposition [31]. The approach in [12] combines sparsification of gradient updates with quantization for reducing the client-to-server communication cost. Model pruning technique has been used for communicating smaller networks, for example by applying the lottery ticket hypothesis for pruning [10, 13]. The work in [17] partitions the model into local parameters for representation learning and global parameters for broadcasting by training the whole model locally. Significant savings is not expected from models having fully connected (FC) layers at the bottom as they possess most of the parameters of the entire model. Moreover, for testing new client data, the scheme needs to pass the data through all local models and ensemble the outputs, which is not suitable in practice. Some of the works avoid communicating the entire model by replacing it with either logits from model outputs as in [9, 22] or binary masks as in [14].

The above schemes mostly improve communication efficiency but do not deal directly with possible computation savings at the client side. To achieve this objective, the typical approach used is mainly focused on broadcasting a sub-network to clients. For example, federated dropout [3] and adaptive federated dropout [1] randomly drop out a fixed percentage of units or filters to broadcast a sub-network combined with lossy compression at each round. Unlike the other knowledge distillation based methods, the approach in [7] trains small models on clients and transfers their knowledge to a large server-side model. The approach used in [14] only communicates and learns personalized binary masks, while freezing the model parameters. This idea, however, restricts the model to update its weights. On the other hand, the method described in HeteroFL [5] broadcasts submodels to clients based on their computation capability assuming that this knowledge is present to the server. All the discussed methods consider a single global model throughout the entire training. We just found out that a concurrently developed method in [26] progressively adds randomly initialized layers (or blocks) from the final model architecture with new classification layers at specific intervals. We expect this scheme to result in a drop of performance at the beginning of each stage and hence, we believe its performance will be significantly lower than ours.

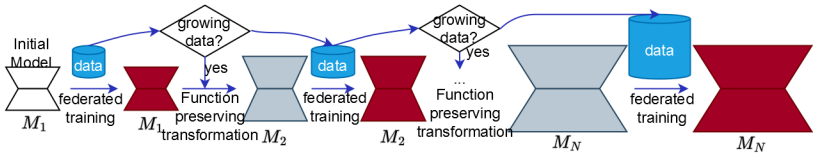


Fig. 1: Overview of the FedNet2Net training.

3 Proposed Approach

We develop a modified training scheme to make the local training computationally much less demanding and reduce the communication requirement while maintaining accuracy. We observe that to achieve a state-of-the-art accuracy, a model needs to capture the essential characteristics of the global data held in the clients, which are known only after the participation of a large number of clients. However, it is not necessary to broadcast such a model throughout the entire training. At the beginning, the model starts with a small amount of data available at relatively few clients, and then progresses with additional information after each round of FL. Intuitively, a model with much lower complexity can be deployed at the beginning and then, we can exploit transfer learning strategies to transfer the knowledge from a lower complexity model to a higher complexity model. In this way, the complexity of the model grows as the data complexity increases, until we reach a final model that can capture the global data.

Our approach is based on transfer learning, but in a different way than standard transfer learning in which the top layers of a student network are directly copied from a teacher network [29]. Instead, the strategy of function-preserving transformations introduced in [4] is utilized, where the student network is initialized in such a way that it represents the same function as the teacher but with different parameterization. We start federated training with a small network. As the training proceeds and more data from clients are exposed, the model is enlarged using two transformations. The first is Net2Widernet, which replaces a model with an equivalent model that is wider, i.e. the student model will have a larger number of units at a certain layer(s). The second operation is Net2DeeperNet, which replaces a model with an equivalent deeper model. These two transformations constitute the essential components of Net2Net. Each transformation preserves the function of the network. After initializing the student network that has the same function as the teacher network, the student network is trained further to improve performance, and once no improvement is detected, the network is enlarged using the Net2Net transformations. We call our scheme FedNet2Net after the Net2Net methodology. Details are presented next.

Let us assume model 1 to model N, $\{M_1, M_2, ..., M_N\}$ are of increasing complexity, where M_1 is a model having the fewest number of layers and units. M_N is a model that can capture the entire data fairly well, and could have the same architecture as deployed in the standard federated training. M_1 through M_N are designed in such a way that for two consecutive models M_i and M_{i+1},

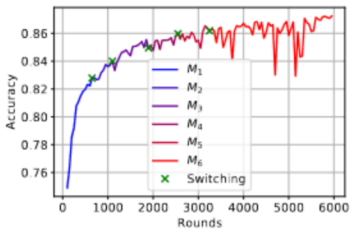


Fig. 2: Model switching in FedNet2Net training using the EMNIST dataset

the architecture of M_{i+1} has either increased the number of layers or increased the number of units in some layer(s) or both compared to M_i, as enlarging the model width or depth is proven to be effective in improving accuracy [7]. Fig. 1 presents the overview of FedNet2Net training. During each stage, the model is deployed after enlarging it by applying a functional preserving transformation to meet the growing data complexity without degradation in performance. In this way, we continue to train a model until its progress saturates after which we switch to the next model. Fig. 2 shows the model switching in FedNet2Net training. We observe that since the highest complexity model is broadcast only at the final stage, we achieve a large reduction in communication bandwidth and clients' computational requirements while achieving similar performance as in the standard FL. In a federated setting, FedNet2Net can be a life long learner and can adapt over a long period of time [24].

We now describe how the transformation is function preserving and how to apply the transformation to transfer knowledge from M_i to M_{i+1}. The student network M_{i+1} has either increased the number of layers or increased the number of units in some layer(s) compared to M_i. In the case of Net2DeeperNet, to insert a FC layer to M_i, we initialize the weights of the inserted layer as the identity matrix. To insert a convolutional layer, we initialize the kernel as the identity filter. Similarly, Net2WiderNet operation replaces a layer with a wider layer while preserving the function value, that is more units for FC layers, or more filters for convolutional layers. For a convolutional kernel K_l of shape (w_l, h_l, i_l, o_l), w_l and h_l denote filter width and height, and i_l, o_l are the number of input and output channels of layer l. To widen this kernel $\hat{K}_l$ having a shape of $(w_l, h_l, i_l, \hat{o}_l)$, where $\hat{o}_l > o_l$, a random mapping is considered as follows:

$$G_l(j) = \begin{cases} j, & \text{if } 1 \leq j \leq o_l \\ \text{random sampling from } \{1, 2, ..., o_l\}, & \text{if } o_l \leq j \leq \hat{o}_l \end{cases}$$

Then the new kernel becomes $\hat{K}_l[x, y, i, j] = K_l[x, y, i, G_l(j)]$. Therefore, the first o_l number of output channels are directly copied while the rest of the output channels are randomly sampled and copied to the new kernel. For preserving the function value, the next layer kernel K_{l+1} is also reduced due to the replication in its input. The new kernel $\hat{K}_{l+1}$ is given as $\hat{K}_{l+1}[x, y, j, k] = \frac{K_{l+1}[x,y,G_l(j),k]}{|\{z|G_l(z)=G_l(j)\}|}$. Similar transformation can be realized for FC layers. More details regarding

the transformations appear in [4]. This operation preserves the function value, that is, for any sample, the function value of the model before and after the transformation remains the same. As we have dropout layers in our models, no extra noise adding is needed for advancing the training since the dropout technique achieves the same goal.

We now address the issue of when to switch to a larger model.

Switching Policy. For the server to decide when to switch from one model to the next, we adopt a switching policy based on the model loss. As the loss at each round fluctuates significantly due to training and random selection of clients, we calculate the window-based loss at t-th round and then compare it to another window of a certain time lag L as follows.

$$S_t = \frac{1}{N}\sum_{i=1}^{N} \text{loss}[t-i-L] - \frac{1}{N}\sum_{i=1}^{N} \text{loss}[t-i] \tag{1}$$

where loss$[i]$ is the weighted average of local model losses from clients at i-th round, L is the time lag between two consecutive windows of size N. In other words, we compute the running average of losses for N rounds of the earlier past and the recent past and take the difference. This measure captures the progress in training based on the loss over the time window. If S_t is not above a certain threshold, then switching to a larger model is performed. Given that the clients are selected randomly during each round, our measure to use the average loss over a window stabilizes the switching decision process. A new model is trained for at least $N + L$ rounds before determining the next switching decision. The thresholds are tuned to get the best possible results. A similar switching policy can be realized by using the validation accuracy evaluated at the server; however this assumes that there is a public validation dataset available at the server.

4 Datasets and Detailed Model Implementations

In this section, we present brief descriptions of the datasets and the detailed model sequences used to conduct various experiments.

4.1 Data Description

In order to evaluate the proposed training approach, three benchmark datasets namely, EMNIST, CIFAR-10, and MNIST are used. For federated training, the training sets in MNIST and CIFAR10 are randomly divided into 100 clients. The EMNIST dataset is an extension of the MNIST, which consists of 671585 images of 62 classes split into 3400 unbalanced non-IID clients [2].

4.2 Performance Evaluation

To evaluate FedNet2Net, we implement and compare the following five methods.

1. ***FedAvg:*** Federated averaging [19] with traditional dropout layer. Here, we add a dropout layer [25] after each convolutional and FC layer. Dropping out of units or filters is performed inside clients.
2. ***FD:*** Federated dropout [3]. Because of broadcasting smaller subnetworks, some amount of communication and computation are saved compared to FedAvg. We consider FedAvg and FD as the baseline methods.
3. ***HeteroFL*** [5]: We uniformly sample computation level of each client at each round from 5 different levels.
4. ***FedNet2Net (FNN):*** This is our approach with traditional dropout layer.
5. ***FedNet2Net-FD (FNN-FD):*** Our approach is combined with federated dropout to reduce further communication and computation. Here, dropout is applied to all models except the smaller ones.

For performance evaluation, we plot the accuracy against total communication of all methods and the reduction in average communication per round versus the accuracy of our approaches over the baselines. The second plot can be used to determine the communication saved for a desired accuracy, and to determine the number of clients that can participate at a round. While not mentioned explicitly, similar reduction in the amount of computations is achieved since the models used are much smaller for the majority of the rounds (if not all). All methods are trained for 6000 rounds and for convenience, the test accuracy is recorded after every 50 rounds.

4.3 Parameters for Switching

As mentioned before, we switch from one model to a larger one when we detect that the training improvement is below a certain threshold over a running window, as described by the equation (1). We use $N = 100$ and $L = 300$ for all datasets. The thresholds for consecutive switching are listed in Table 1. The switching is decided at the server, and hence, no extra computation or communication is incurred at the client side.

4.4 Model Description and Hyper-parameters

The hyperparameters and models for FedAvg, FD and the final stage of FedNet2Net are set as suggested in [3]. The learning rates and the number of clients per round are listed in Table 1. The number of epochs per round is set to 1 and the batch size for local training is 10. SGD optimizer and sparse categorical

Table 1: Thresholds for five consecutive switchings and hyperparameters for three datasets.

dataset	Thresholds for policy (1)	learning rate	clients per round
EMNIST	$[0.08, 0.04, 0.02, 0.01, 0.005]$	0.035	35
CIFAR10	$[0.12, 0.11, 0.10, 0.09, 0.08]$	0.05	10
MNIST	$[0.04, 0.02, 0.01, 0.005, 0.0025]$	0.015	10

Table 2: Consecutive models of FedNet2Net training for the EMNIST and MNIST datasets. The number of units are written in parenthesis. For MNIST, we use the same architectures except each classification layer has 10 units and the number of units in the second last layer is 128 for models 1-4, 256 for model 5 and 512 for model 6. Kernel size is 5×5.

Model number	Architecture	Parameters
model 1	Conv2D(16) → maxpool(4,4) → FC(512) → FC(62)	434K
model 2	**Conv2D(32)** → maxpool(4,4) → FC(512) → FC(62)	836K
model 3	Conv2D(32) → maxpool(2,2) → **Conv2D(32)** → maxpool(2,2) → FC(512) → FC(62)	862K
model 4	Conv2D(32) → maxpool(2,2) → **Conv2D(64)** → maxpool(2,2) → FC(512) → FC(62)	1.7M
model 5	Conv2D(32) → maxpool(2,2) → Conv2D(64) → maxpool(2,2) → **FC(1024)** → FC(62)	3.3M
model 6	Conv2D(32) → maxpool(2,2) → Conv2D(64) → maxpool(2,2) → **FC(2048)** → FC(62)	6.6M

cross-entropy loss are used. Dropout rate is set to 0.125. The sequence of models used in FedNet2Net are presented in Tables 2 and 3 along with the number of parameters in each model. Changes in consecutive models are highlighted. RELU activation and dropout is applied after each convolutional and FC layer except the last classification layer, where a softmax activation is applied.

5 Results

Fig. 3 compares the performance between FedNet2Net and the baseline methods using the model loss based switching policy. The figures on the left present the accuracy against total communication (in bits) with switching positions in our approach. We observe that for any fixed amount of communication, FedNet2Net achieves higher and occasionally equal accuracy compared to FedAvg, federated dropout and HeteroFL; the only exception is the performance of HeteroFL on CIFAR10 at the high accuracy end, which we believe is due to adding batch normalization layers in HeteroFL, which boosted the training. Otherwise, HeteroFL training is highly unstable.

The figures on the right present the percentage reduction in communication per round of FNN over FedAvg, FNN-FD over FD, FD over FedAvg, HeteroFL over FedAvg and FNN-FD over FedAvg. It is observed that, at the slightly lower accuracy regions than the best possible, the percentage savings is approximately more than 90% per round. This saving remains constant for most of the accuracy regions. At the higher accuracy region, the percentage savings starts to drop, which intuitively makes sense as we are switching to larger models to capture the growing data complexity. Moreover, the reduction of FD and HeteroFL over FedAvg remains the same for all accuracies as they broadcast roughly the same

Table 3: Consecutive models in FedNet2Net for CIFAR10. Kernel size is 3×3 everywhere except the last two convolutional layers of model 6, where it is 1×1.

Model number	Architecture	Parameters
model 1	Conv2D(32) → maxpool(3,3) → Conv2D(64) → maxpool(3,3) → Conv2D(10) → GlobalAveragePooling2D (GAP) → FC(10)	20K
model 2	Conv2D(32) → **Conv2D(32)** → maxpool(3,3) → Conv2D(64) → **Conv2D(64)** → maxpool(3,3) → Conv2D(10) → GAP → FC(10)	66K
model 3	**Conv2D(64)** → **Conv2D(64)** → maxpool(3,3) → **Conv2D(128)** → **Conv2D(128)** → maxpool(3,3) → Conv2D(10) → GAP → FC(10)	262K
model 4	**Conv2D(96)** → **Conv2D(96)** → maxpool(3,3) → **Conv2D(192)** → **Conv2D(192)** → maxpool(3,3) → Conv2D(10) → GAP → FC(10)	586K
model 5	Conv2D(96) → Conv2D(96) → maxpool(3,3) → Conv2D(192) → Conv2D(192) → maxpool(3,3) → **Conv2D(192)** → Conv2D(10) → GAP → FC(10)	918K
model 6	Conv2D(96) → Conv2D(96) → maxpool(3,3) → Conv2D(192) → Conv2D(192) → maxpool(3,3) → Conv2D(192) → **Conv2D(192)** → Conv2D(10) → GAP → FC(10)	955K

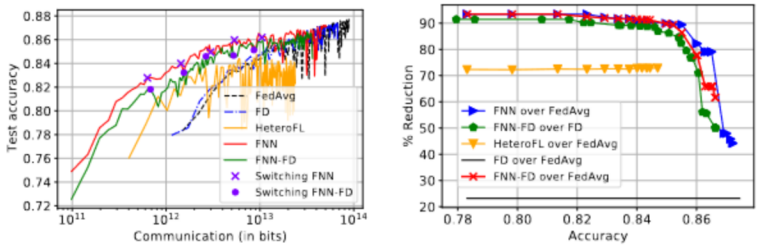
number of parameters at all rounds. However, additional per round communication savings is achieved by combining FedNet2Net with FD. Similar reduction is achieved when we separated a validation set from the training data to measure validation accuracy for making switching decisions. Therefore, our training approach based on model growing reduces significant amount of communication per round. Moreover, as our approach starts from a significantly small model and gets transformed into a higher capacity model, it opens the opportunity to adapt the model as data complexity grows.

Effect of choosing the different set of intermediate models. Given the final model, we implement FedNet2net with a different set of intermediate models to show the effect of choosing them differently. For EMNIST, we implement FedNet2Net with sets of 6 and 8 models (FNN-6 and FNN-8), and the results are presented in Fig. 4. FNN-8 has a smaller starting network than FNN-6. We observe that FNN-8 performs better than FNN-6 in terms of communication reductions. This is intuitive as the final model is broadcast for lesser number of rounds. Similarly for CIFAR10, we implement FedNet2Net by changing some of the intermediate models mentioned in Table 3. The reductions achieved is almost the same as in Fig. 3(b). We omitted the figure due to space limitations.

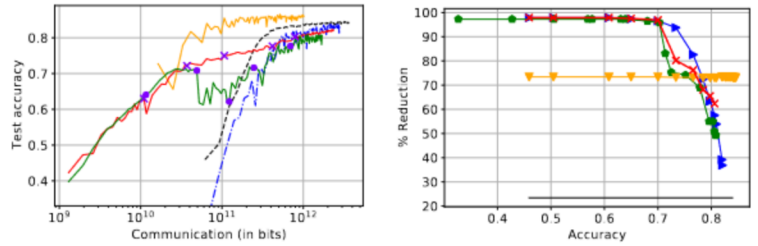
6 Conclusion

In this paper, a new federated training scheme, based on the model growing strategy, is proposed for saving both communication cost and computations at

(a) EMNIST dataset



(b) CIFAR10 dataset



(c) MNIST dataset

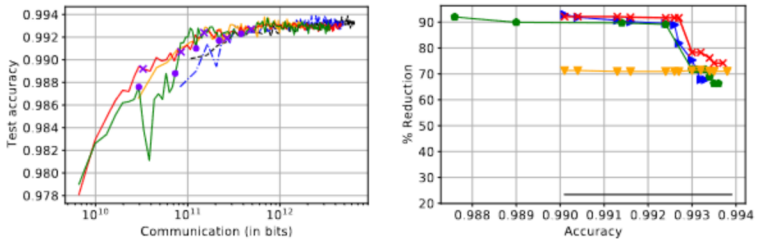


Fig. 3: Accuracy against communication of the FedAvg, FD, HeteroFL, FedNet2Net and FedNet2Net-FD (left), and percentage reduction in communication per round of FedNet2Net, FD and HeteroFL over FedAvg (right). The markings in the left figures indicate the switching positions based on policy (1). Accuracies fluctuate due to training from randomly selected clients at each round.

the clients. At the initial stage, as the model is trained using only a small amount of data, deploying a model with minimal capacity saves both communication and local computation. Next, the efficient switchings to the enlarged models using function-preserving transformations ensure a continuous improvement in performance as long as the complexity of the data increases. In this way, the capacity of the model is increased until the model captures the overall data complexity. The most complex model is deployed only at the final stages of training, and thereby, saving a substantial amount of communication and local computations. Extensive experiments on three benchmarks demonstrate that the proposed Fed-

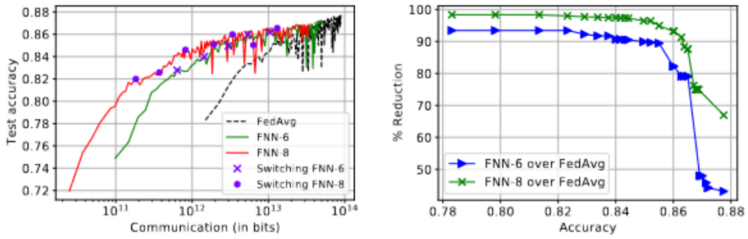


Fig. 4: Performance comparison of FedNet2Net with different set of intermediate models for the EMNIST dataset.

Net2Net training scheme can save significant amount of communication cost and local computations per round.

References

1. Bouacida, N., Hou, J., Zang, H., Liu, X.: Adaptive federated dropout: Improving communication efficiency and generalization for federated learning. arXiv preprint arXiv:2011.04050 (2020)
2. Caldas, S., Duddu, S.M.K., Wu, P., Li, T., et al.: Leaf: A benchmark for federated settings. arXiv preprint arXiv:1812.01097 (2018)
3. Caldas, S., Konečny, J., McMahan, H.B., Talwalkar, A.: Expanding the reach of federated learning by reducing client resource requirements. arXiv preprint arXiv:1812.07210 (2018)
4. Chen, T., Goodfellow, I., Shlens, J.: Net2Net: Accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641 (2015)
5. Diao, E., Ding, J., Tarokh, V.: HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. arXiv preprint arXiv:2010.01264 (2020)
6. Elkordy, A.R., Avestimehr, A.S.: HeteroSAg: Secure aggregation with heterogeneous quantization in federated learning. IEEE Trans. on Communications (2022)
7. He, C., Annavaram, M., Avestimehr, S.: Group knowledge transfer: Federated learning of large CNNs at the edge. Advances in Neural Information Processing Systems (NeurIPS) **33**, 14068–14080 (2020)
8. Imteaj, A., Thakker, U., Wang, S., et al.: A survey on federated learning for resource- constrained IoT devices. IEEE IoT Journal **9**(1), 1–24 (2021)
9. Itahara, S., Nishio, T., Koda, Y., Morikura, M., et al.: Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-IID private data. arXiv preprint arXiv:2008.06180 (2020)
10. Jiang, Y., Wang, S., Valls, V., Ko, B.J., et al.: Model pruning enables efficient federated learning on edge devices. IEEE Trans. on Neural Networks and Learning Systems (NNLS) (2022)
11. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., et al.: Advances and open problems in federated learning. Foundations and Trends in Machine Learning **14**(1–2), 1–210 (2021)
12. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., et al.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)

13. Li, A., Sun, J., Wang, B., Duan, L., et al.: LotteryFL: Empower edge intelligence with personalized and communication-efficient federated learning. In: IEEE/ACM Symposium on Edge Computing. pp. 68–79 (2021)
14. Li, A., Sun, J., Zeng, X., Zhang, M., et al.: FedMask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In: 19th ACM Conf. on Embedded Networked Sensor Systems. pp. 42–55 (2021)
15. Li, C., Li, G., Varshney, P.K.: Communication-efficient federated learning based on compressed sensing. IEEE Internet of Things Journal **8**(20), 15531–15541 (2021)
16. Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine **37**(3), 50–60 (2020)
17. Liang, P.P., Liu, T., Ziyin, L., Salakhutdinov, R., et al.: Think locally, act globally: Federated learning with local and global representations. arXiv preprint arXiv:2001.01523 (2020)
18. Mao, Y., Zhao, Z., Yan, G., Liu, Y., et al.: Communication efficient federated learning with adaptive quantization. ACM Trans. on Intelligent Systems and Technology (2021)
19. McMahan, B., Moore, E., Ramage, D., Hampson, S., et al.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. pp. 1273–1282 (2017)
20. Qiao, Z., Yu, X., Zhang, J., Letaief, K.B.: Communication-efficient federated learning with dual-side low-rank compression. arXiv preprint arXiv:2104.12416 (2021)
21. Rothchild, D., Panda, A., Ullah, E., Ivkin, N., et al.: FetchSGD: Communication-efficient federated learning with sketching. In: 37th International Conference on Machine Learning. vol. 119, pp. 8253–8265 (2020)
22. Sattler, F., Marban, A., Rischke, R., Samek, W.: Communication-efficient federated distillation. arXiv preprint arXiv:2012.00632 (2020)
23. Savazzi, S., Nicoli, M., Rampa, V.: Federated learning with cooperating devices: A consensus approach for massive IoT networks. IEEE Internet of Things Journal **7**(5), 4641–4654 (2020)
24. Silver, D.L., Yang, Q., Li, L.: Lifelong machine learning systems: Beyond learning algorithms. In: 2013 AAAI Spring Symposium Series (2013)
25. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., et al.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
26. Wang, H.P., Stich, S.U., He, Y., Fritz, M.: ProgFed: Effective, communication, and computation efficient federated learning by progressive training. In: International Conference on Machine Learning. pp. 23034–23054 (2022)
27. Wu, C., Wu, F., Lyu, L., Huang, Y., et al.: Communication-efficient federated learning via knowledge distillation. Nature communications **13**(1), 1–8 (2022)
28. Xu, J., Du, W., Jin, Y., He, W., et al.: Ternary compression for communication-efficient federated learning. IEEE Trans. on NNLS (2020)
29. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in NeurIPS. vol. 27 (2014)
30. Yu, R., Li, P.: Toward resource-efficient federated learning in mobile edge computing. IEEE Network **35**(1), 148–155 (2021)
31. Zheng, H., Gao, M., Chen, Z., Feng, X.: A distributed hierarchical deep computation model for federated learning in edge computing. IEEE Transactions on Industrial Informatics **17**(12), 7946–7956 (2021)