IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 29, 2020, accepted February 10, 2021, date of publication March 2, 2021, date of current version March 12, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3063291

Accelerating Federated Learning for IoT in Big Data Analytics With Pruning, Quantization and Selective Updating

WENYUAN XU[1], WEIWEI FANG[1,2], YI DING[3], MEIXIA ZOU[1], AND NAIXUE XIONG[4], (Senior Member, IEEE)

[1]School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China
[2]Key Laboratory of Industrial Internet of Things & Networked Control, Ministry of Education, Chongqing 400065, China
[3]School of Information, Beijing Wuzi University, Beijing 101149, China
[4]College of Intelligence and Computing, Tianjin University, Tianjin 300350, China

Corresponding author: Weiwei Fang (fangww@bjtu.edu.cn)

This work was supported in part by the Beijing Municipal Natural Science Foundation under Grant L191019, in part by the Open Project of Key Laboratory of Industrial Internet of Things & Networked Control, Ministry of Education under Grant 2019FF03, in part by the Open Project of Beijing Intelligent Logistics System Collaborative Innovation Center under Grant BILSCIC-2019KF-10, in part by the CERNET Innovation Project under Grant NGII20190308, in part by the Traffic Control Technology Innovation Fund under Grant 9907006515, in part by the Research Base Project of Beijing Municipal Social Science Foundation under Grant 18JDGLB026, and in part by the Science and Technique General Program of Beijing Municipal Commission of Education under Grant KM201910037003.

ABSTRACT The ever-increasing number of Internet of Things (IoT) devices are continuously generating huge masses of data, but the current cloud-centric approach for IoT big data analysis has raised public concerns on both data privacy and network cost. Federated learning (FL) recently emerges as a promising technique to accommodate these concerns, by means of learning a global model by aggregating local updates from multiple devices without sharing the privacy-sensitive data. However, IoT devices usually have constrained computation resources and poor network connections, making it infeasible or very slow to train deep neural networks (DNNs) by following the FL pattern. To address this problem, we propose a new efficient FL framework called FL-PQSU in this paper. It is composed of 3-stage pipeline: structured pruning, weight quantization and selective updating, that work together to reduce the costs of computation, storage, and communication to accelerate the FL training process. We study FL-PQSU using popular DNN models (AlexNet, VGG16) and publicly available datasets (MNIST, CIFAR10), and demonstrate that it can well control the training overhead while still guaranteeing the learning performance.

INDEX TERMS Federated learning, Internet of Things, big data, model compression, network pruning.

I. INTRODUCTION

Over the past decades, Artificial Intelligence (AI) technology has made great strides in a variety of real-life applications, ranging from image recognition, video surveillance, speech synthesis, to machine translation. The development of these AI-based applications relies heavily on the knowledge embedded in big data, which are indispensable for training high-performance AI models such as the deep neural networks (DNNs). The Internet of Things (IoT) have brought about an explosion in the availability and quantity of data for achieving such purposes, with an estimated 50 billion devices connected to the Internet by 2020 [1]. In current implementations, the real-time data collected by IoT devices are uploaded to and processed by a cloud server in a centralized fashion. However, the cloud-based model training raises public concern about the sharing of privacy-sensitive data, and may suffer from unacceptable long latency as well as high traffic burden [2]. To mitigate these challenges, it is preferable to decouple model training from the need for remotely collecting and centrally processing of the raw data.

In view of these shortcomings of the cloud-centric solution, researchers have recently proposed a new approach to train a shared global model on datasets decentrally located at a loose federation of participating devices (clients), termed as Federated Learning (FL) [3]. In FL, each client computes an update to the global model based on its local training dataset, and then uploads the updated local model parameters to a FL

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaoqing Pan.

server. The FL server aggregates the updates from all clients to improve the model, and sends the newly updated global model back to the clients. The steps will be repeated for a number of rounds until the convergence or a desired accuracy is reached [2]. As the training data is kept locally at each client and never shared with others, the FL approach provides a very promising solution to privacy-preserving and resource-efficient big data analytics.

Nevertheless, a critical challenge posed to FL is the training overhead. On one hand, popular DNN models usually contain from tens of thousands of to hundreds of millions of parameters [4]. The ever-increasing network complexity and training data make the model training very compute- and memory-intensive [2]. On the other hand, the high-dimensional and frequent parameter updates incur high communication costs and may result in the training bottleneck [5]. It is even worse that the IoT devices are usually resource constrained in terms of computation capability and communication bandwidth, so the training time will be too long or even unbearable. This is not conducive to the rapid deployment and application of AI models in practice. Though a lot of model compression techniques have been proposed for efficient processing of DNNs, most solutions are designed to accelerate the DNN inference processing [4], [6], and few work has taken the training costs into account [7].

In this paper, we propose FL-PQSU, a training acceleration framework that compresses and optimizes DNN models during the FL training process. The processing pipeline, which runs for a specific model learning task, consists of three stages: structured pruning, weight quantization and selective updating. The pruning stage alleviates the computation overhead by reducing the number of parameters and FLOPs, while the latter two stages mitigate the communication overhead by reducing the total sizes of transmission and the total times of update, respectively. To the best of our knowledge, FL-PQSU represents the first effort to unify the above three mechanisms into one framework. It can be further optimized for different specific resource limitations or performance requirements. For example, the user can freely determine which mechanism among the three to be used in FL training, according to the target model and dataset [7]. In all, the contributions of our work are summarized as follows.

- We propose FL-PQSU, a new framework that enables efficient model training on resource-limited IoT devices, by incorporating overhead reduction techniques into the standard FL.
- We design the optimization approaches targeting for reduction of the computation and communication overheads respectively. On one hand, we perform one-shot structured pruning to the initial model at the FL server, so as to reducing the model size and the computation overhead. On the other hand, we adopt two approaches to reduce the communication overhead for model updates, i.e., we not only quantize model weights to reduce the total transferred bits, but also preclude helpless updates from being uploaded to reduce the total update times.
- We conduct extensive experiments to evaluate the performance of our FL-PQSU using popular DNN models (i.e., AlexNet, VGG16) and publicly available datasets (i.e., MNIST, CIFAR10), which confirm that FL-PQSU enables a FL system to dramatically reduce the computation and communication overheads in model training, while preserving the model accuracy at a desired level.

The remainder of this paper is structured as follows. In Section II, we outline the related work in the fields of efficient FL and DNN model compression, respectively. Section III describes our proposed FL-PQSU framework in detail. Experimental results are analyzed in Section IV. Finally, we conclude our work in Section V.

II. RELATED WORK

In this section, we outline several important prior works on efficient FL and DNN model compression. Due to the space limitation, interested readers can refer to [2], [4] for more detailed and comprehensive introductions on these two research topics.

A. EFFICIENT FEDERATED LEARNING

FL, proposed by McMahan *et al*. [3], aims to solve the problem of decentralized training over distributed data sources without direct access to the privacy-sensitive data. The Federated Averaging (FedAvg) algorithm introduced in [3] has become the state-of-the-art method for FL. Considering that high training costs remain as the main obstacle hindering federated training, many research efforts have been made in recent years.

Many studies focus on reducing the communication costs between clients and the FL server during the training process. The concept of structured and sketched updates is proposed in [8] to reduce the size of uploaded model updates during training. However, it only takes the optimization of client-to-server communication into account, and may come with no guarantee of convergence [9]. In contrast, Caldas *et al*. [10] proposed to optimize server-to-client communications costs, by employing federated dropout to train small sub-models on clients and lossy compression to server-to-client exchanges. While these two studies dedicate to decrease the transferred bits during model training, some other works attempt to preclude unimportant updates from being uploaded for mitigating communication overhead [5], [9]. In the edge Stochastic Gradient Descent (eSGD) algorithm [5], only a small portion of important gradients are selected to be updated to the server in each communication round. However, eSGD suffers from non-negligible accuracy loss, and its performance fluctuates arbitrarily with the hyperparameters used. Similar to eSGD, the Communication Mitigated Federated Learning (CMFL) algorithm [9] merely uploads relevant updates to the server. It not only reduces the communication overhead in FL training, but also provides provable convergence guarantees.

More recently, model compression has been revisited as a method for cost reduction in FL training [7]. By jointly training and pruning the DNN model in a federated manner, the approach proposed in [7] can well reduce the model size, resulting in significant decrements in both communication time and computation time. However, the weight pruning method adopted in [7] is considered harmful [11], due to its inferiority in computation efficiency and storage overhead. What's more, the work lacks consideration of communication overheads during the updating process. This problem will be more salient when the FL training involves a large number of clients [9].

B. DNN MODEL COMPRESSION

Modern DNN models are known to be severely resource intensive, and various compression techniques have already been proposed to reduce the model size as well as the number of operations [6]. Existing studies can be roughly categorized into five solution families, namely network pruning, data quantization, knowledge distillation, low-rank factorization and compact network design [4]. Here, we just briefly summarize the first two techniques that are related to this work.

Network pruning targets at removing redundant structures and parameters from the DNN model to accelerate the inference. Early works [6], [12] proposed to remove the connections with small-weights whose values are below a given threshold. However, weight pruning inherently induces unstructured sparsity, and cannot function without dedicated hardware and libraries [11]. The following works focus more on the structured pruning techniques, especially channel pruning. For example, Li *et al.* uses the ℓ_1-norms of per-channel weights to select and prune redundant channels [13]. Network slimming [14] imposes sparsity regularization on the scaling factors in batch normalization layers during training, so as to identify and prune insignificant channels. Unlike these previous approaches, Play and Prune [15] allows to specify the error tolerance limit instead of the pruning ratio for each layer. Wang *et al.* [16] verify that pruning from randomly initialized weights directly can result in more diverse pruned structures with competitive performance. More recent studies [7], [17] proposed to prune the model during training for not only improving the performance of inference but also reducing the costs of training.

Data quantization aims to represent weights or activations using lower-precision numerical formats rather than the 32-bit floating point (FP32), in order to reduce the model size and improve the computing efficiency. BinaryConnect [18] trains a DNN with binary weights (i.e., -1 or 1) during propagations, and replaces resource-hungry multiply-accumulate operations by simple additions and subtractions. By leveraging an extra 0 state in DNN weights, Ternary Weight Network (TWN) [19] can achieve a good trade-off between model size and inference accuracy. However, such binary or ternary quantizations are challenged by the non-negligible accuracy degradation, particularly for large-scale models. The

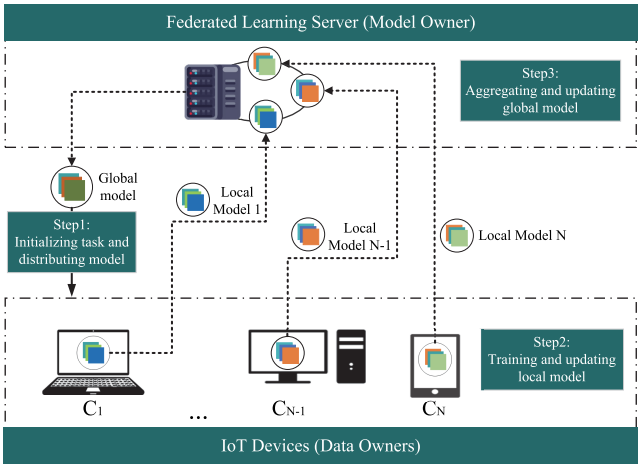


FIGURE 1. An overview of FL training process for IoT in big data analytics.

Stochastic Quantization (SQ) algorithm [20] is proposed to overcome this accuracy drop problem, by stochastically selecting a portion of weights to quantize and gradually increasing the SQ ratio to quantize the whole network. Actually, it has been revealed that quantizing weights and activations using 8-bit integers (INT8) can result in a relatively low loss of accuracy [6], [21]. Thus, INT8 quantization is now widely supported by deep learning frameworks, e.g., Google TensorFlow, Nvidia Tensor RT, and Baidu PaddlePaddle [22].

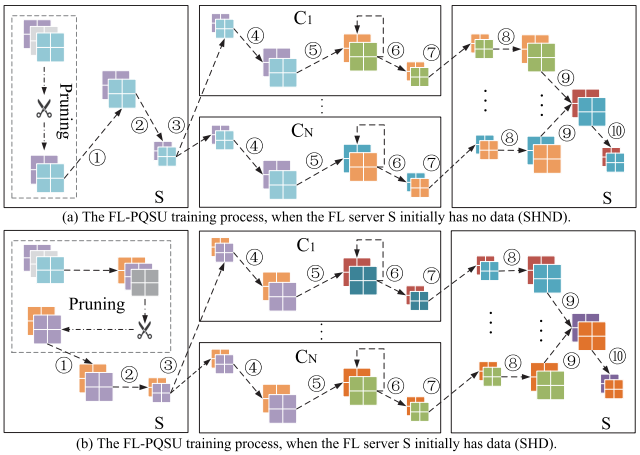
III. OUR PROPOSED FL-PQSU FRAMEWORK

A. STANDARD FL PROCEDURE

A typical FL system consists of a FL server S and a set $\mathcal{C}$ ($|\mathcal{C}| \geq 1$) of participating clients, each of which owns a private dataset $d_{c\in\mathcal{C}}$. Each client c uses its local dataset d_c to train a local model $\mathbf{m}_c$, and then sends the local parameters as an update to S. The FL server S collects all the local models $\mathbf{m} = \cup_{c\in\mathcal{C}}\mathbf{m}_c$, and finally obtains the global model $\mathbf{M}_G$ according to some aggregation rule [2]. Note that this is different from the conventional cloud-centric training, which trains the model $\mathbf{M}_G$ by centrally aggregating and processing the data from all clients $\mathbf{D} = \cup_{c\in\mathcal{C}}d_c$.

As shown in Fig. 1, the training process of FL consists of the following three steps:

- *Step 1 (Initializing task and distributing model)*: In round 0, S determines the specific training task, including the target model, the data requirements, and the hyperparameters (e.g., batch size). Then, it broadcasts the initial global model $\mathbf{M}_G^0$ and task settings to all participating clients.
- *Step 2 (Training and updating local model)*: In round t, based on the global model $\mathbf{M}_G^t$, client c updates the local model parameters $\mathbf{m}_c^t$ using its local data. Specifically, it aims to obtain optimal parameters $\mathbf{m}_c^t$ that minimize the loss function $L(\mathbf{m}_c^t)$. Then, client c uploads the updated local parameters to S.
- *Step 3 (Aggregating and updating global model)*: In round t, S aggregates all the received local models, with



(a) The FL-PQSU training process, when the FL server S initially has no data (SHND).

(b) The FL-PQSU training process, when the FL server S initially has data (SHD).

FIGURE 2. An overview of FL-PQSU training under different initial conditions.

the objective to minimize the global loss function [3]:

$$L(\mathbf{M}_c^t) = \frac{1}{|\mathcal{C}|}\sum_{c=1}^{|\mathcal{C}|} L(\mathbf{m}_c^t). \tag{1}$$

Then, S will broadcast the updated global model $\mathbf{M}_G^{t+1}$ to the clients for the training in next round $t+1$, until the convergence of $L(\mathbf{M}_c^t)$ or the achievement of a desired accuracy.

As discussed in Section II, in the IoT application scenario where clients usually have constrained computation resources and poor network connections, the above FL training process is very costly on computation and communication, especially when training a complex DNN model [7], [17]. As such, it is very imperative to reduce the overheads on computation and communication at clients.

B. FL-PQSU FRAMEWORK

Our goal is to reduce the computation and communication overheads to accelerate the FL training process. To achieve this goal, we propose the FL-PQSU framework, a three-stage pipeline to reduce the training costs while still preserving the prediction accuracy at a desired level. First, the pruning technique is used to reduce the model size and the computation cost, by removing redundant parts and keeping informative parts. Next, data quantization is applied to reduce the total bits transferred in the model updates (i.e., local and global). Finally, we propose selective updating to avoid unnecessary local model updates from some clients, so as to reduce the total number of updates in training.

The FL-PQSU framework integrates the above three techniques, namely Pruning, Quantization, and Selective Updating, into the original FL training process in Fig. 1. The new training process is shown in Fig. 2, and the description of our extensions to the standard FL are described as follows:

- *Step 1 (Initializing task and distributing model ①②③)*: In round 0, S initializes the model parameters, and performs pruning to the global model $\mathbf{M}_G^0$. Note that the model pruning is one-shot, i.e., it is only performed in round 0. After quantizing the weights of pruned model $\mathbf{M}_{GP}^0$, S broadcasts the compressed model $\mathbf{M}_{GPQ}^0$ to the clients.
- *Step 2 (Training and updating local model ④⑤⑥⑦)*: In round t, client c dequantizes $\mathbf{M}_{GPQ}^t$, and performs the local training. Based on the variation of its loss values, c determines whether to uploads its local update $\mathbf{m}_c^t$ to S. If so, c will quantize the model weights of $\mathbf{m}_c^t$, and uploads the quantized local parameters to S.
- *Step 3 (Aggregating and updating global model ⑧⑨⑩)*: In round t, S dequantizes all the received local models, and then aggregates them to update the global model $\mathbf{M}_{GP}^{t+1}$. After quantization, S will broadcast the compressed model $\mathbf{M}_{GPQ}^{t+1}$ to the clients.

It is worth noting that initial model pruning can be carried out in two different cases with regard to data availability at the FL server.

- *Server has no data (SHND)*: In this case, S is a dedicated FL server with no training data. This is the most common case in FL [3]. Fortunately, according to [16], the model pruning can be directly started from randomly initialized weights, and model convergence is guaranteed.
- *Server has data (SHD)*: In this case, S itself has kept a certain amount of data samples, which could be obtained from either the data sharing of some clients or the data collection on its own. Actually, the initial model training at S before the FL process starts will help to further accelerate the convergence of global model.

C. STRUCTURED PRUNING

Modern DNN models generally have very large models with a huge number of weights. Due to the resource limitations, training DNN models on the IoT device is usually a time-consuming task [7]. The most recent studies [7], [17] have found that, though model pruning is mainly proposed to improve the inference performance, it can also significantly accelerate training by reducing model size and computation burden. Note that the pruning should be conducted based on some predefined metric, because shrinking the model size blindly may result in non-negligible accuracy loss. In this work, we perform ℓ_1-norm based, one-shot channel pruning in FL-PQSU for training optimization. Our choice is based on the following three reasons: (1) Other than weight pruning [7], channel pruning produces hardware-friendly models without introducing irregular sparsity [11]. (2) ℓ_1-norm can be easily calculated for measuring the importance of filters [13], while most pruning criteria [14], [15] can only be obtained in the formal training process. (3) Unlike [7], only one-shot pruning is adopted by FL-PQSU, as the further pruning in federated training incurs additional overhead, but contributes little to performance improvement [13]. According to our testing experiments, ℓ_1-norm based pruning outperforms blind model shrinking by about 1% accuracy loss for large models like VGG16, which can't be simply neglected [16].

In FL-PQSU, we can prune not only convolutional layers but also fully connected layers, if necessary. That's because convolutional layers contribute to the majority of computation overhead, while fully connected layers may contribute to the storage space of mode size [13], [15]. The ℓ_1-norm based pruning algorithm is shown in Algorithm 1. For the i-th convolutional layer, let C_{in}^i and C_{out}^i denote the input and out channels, H_{in}^i and H_{out}^i denote the number of input and output channels, H_{in}^i/W_{in}^i and H_{out}^i/W_{out}^i denote the height/width of the input and output feature maps, and $\mathcal{K} \in \mathbb{R}^{k\times k}$ (e.g., 3×3) be the 2D kernel, respectively. By applying C_{out}^i 3D filters $\mathcal{F}_{i,j} \in \mathbb{R}^{C_{in}^i \times k \times k}$ to the C_{in}^i input channels, the input feature maps $\mathbf{x}_{in}^i \in \mathbb{R}^{C_{in}^i \times H_{in}^i \times W_{in}^i}$ can be transformed into the output feature maps $\mathbf{x}_{out}^i \in \mathbb{R}^{C_{out}^i \times H_{out}^i \times W_{out}^i}$. By removing a filter from the i-th convolutional layer, we can reduce totally $C_{in}^i \times k^2 \times H_{out}^i \times W_{out}^i$ operations and $C_{in}^i \times k^2$ weights [13]. Meanwhile, we also prune the redundant neurons from fully connected layers [15]. For the k-th fully connected layer, let n_k denote the number of neurons, and $w_k \in \mathbb{R}^{n_k \times n_{k+1}}$ denote the number of weights. By removing a neuron from the k-th fully connected layer, we can reduce totally n_{k+1} operations and n_{k+1} weights.

Algorithm 1 ℓ_1-Norm Based Pruning Algorithm

Input:
M_{init}: the initial DNN model
I: the total number of convolutional layer
K: the total number of fully connected layer
Output:
M_{pruned}: the pruned DNN model

Procedure
if Server has data **then**
  Use the available data samples to train M_{init}
else
  Assign randomly initialized weights to M_{init}
end if
for each layer $i \in \{1, \ldots, I\}$ **do**
  For each filter $\mathcal{F}_{i,j}$, calculate ℓ_1-norm $s_j = \|\mathcal{F}_{i,j}\|_1$
  Sort the filters by s_j
  Prune a certain number of smallest filters
  Copy the remaining filters to the new model M_{pruned}
end for
for each layer $k \in \{1, \ldots, K\}$ **do**
  For each weight $w_{k,j}$, calculate ℓ_1-norm $s'_j = \|w_{k,j}\|_1$
  Sort the weights by s'_j
  Prune a certain number of smallest weights
  Copy the remaining weights to the new model M_{pruned}
end for
return M_{pruned}

D. WEIGHT QUANTIZATION

Though network pruning can help to reduce computation overhead and model size, the pruned model may still have a large number of parameters. For example, VGG16 has 138M weights in total, and 13.8M weights are still remained after 90% weights are removed. In such cases, communication has become a severe bottleneck, especially when the network bandwidth is low. Quantization can help reduce bandwidth and computation requirements, by replacing the full precision 32-bit floating point numerical format with a lower bit-width format, e.g., INT8. However, it has been revealed that training DNN models with low precision directly may result in non-negligible accuracy loss [22]. Therefore, in FL-PQSU we still use full precision parameters in model training, but quantize and dequantize the transfered data in model updates.

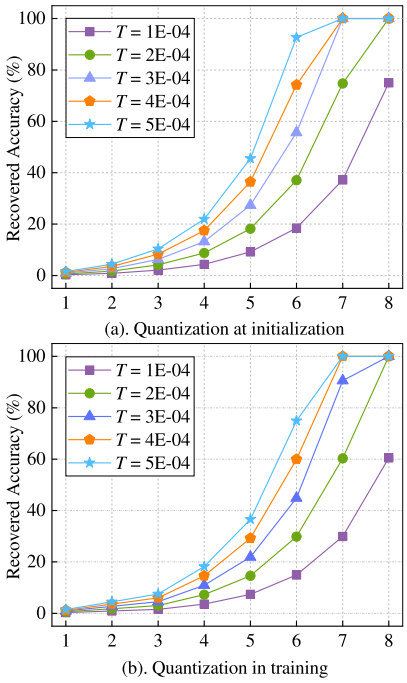


FIGURE 3. The impact of quantization on accuracy using different bit-widths.

The choice of bit-width for data representation has a great impact on accuracy recovery. We conduct a simple quantization experiment on the second convolutional layer of AlexNet, using different bit-widths ranging from 1 to 8 to quantize the layer weights. Fig. 3 shows the results on accuracy recovery at initialization and after training, respectively. Note that T represents the tolerable difference between the original value and the dequantized value, less than which these two values can be treated as equal. The results confirm the superiority of INT8 in accuracy as compared to the other options [21]. Thus, regarding the generality and generalization of our solution, the INT8-based quantization is adopted by FL-PQSU.

Let X_{sf} denotes the scaling factor, Z_q denotes the zero point in quantized values, $W_f \in [W_f^{min}, W_f^{max}]$ and $W_i \in [Q_{min}, Q_{max}]$ denote the float and integer weights before and after quantization, respectively. Specifically, X_{sf} and Z_q can be given as:

$$X_{sf} = \frac{W_f^{max} - W_f^{min}}{Q_{max} - Q_{min}}. \tag{2}$$

$$Z_q = \begin{cases} Q_{max}, \\ \quad W_f^{max}/X_{sf} \in (-\infty, 0) \\ Q_{max} - W_f^{max}/X_{sf}, \\ \quad W_f^{max}/X_{sf} \in [0, Q_{max} - Q_{min}] \\ Q_{min}, \\ \quad W_f^{max}/X_{sf} \in (Q_{max} - Q_{min}, +\infty) \end{cases}. \quad (3)$$

A weight can be quantized from FP32 to INT8 according to:

$$W_i = \lfloor Z_q + W_f/X_{sf} \rceil. \quad (4)$$

Conversely, W_i can be dequantized back into the FP32 format, W_f', according to:

$$W_f' = X_{sf}(W_i - Z_q). \quad (5)$$

E. SELECTIVE UPDATING

The FL training generally involves a number of participating clients, resulting in considerable communication overhead because of frequent updating exchanges and low link bandwidth between the FL server and clients. However, it has been found that not all client-side updates are relevant enough to model improvement [5], [9], as they may be trained over biased or device-specific data samples. Therefore, it is necessary to preclude helpless updates from being uploaded, so as to avoid unnecessary data transfer and high communication cost.

In FL-PQSU, the loss value is used by the client to judge the importance of current update [5]. Intuitively, the loss function is anticipated to be converged in each round, i.e., the loss value should be gradually reduced and closer to the optimal value. However, the loss value often fluctuates erratically in the training process. In our design, each client records the loss value l_{last} in the last upload, and compares it with the current value $l_{current}$. If $l_{last} > l_{current}$, this indicates that the current update is beneficial to the loss function and should be uploaded to the FL server. Otherwise, this indicates that in current round $l_{current}$ is undesirable. Accordingly, the current update are prevented by the client to be uploaded, and the FL server will have to use the latest update of this client instead.

IV. PERFORMANCE EVALUATION

A. EXPERIMENT SETTINGS

1) DATASETS

We conduct experiments on image classification tasks using the MNIST and CIFAR10 datasets, which are widely used for benchmarking in the FL researches [3], [8], [10]. The MNIST dataset contains 60000 training and 10000 testing gray-scale images of handwritten digits, and the CIFAR10 dataset contains 50000 training and 10000 testing colored images in 10 classes.

2) MODELS

For MNIST, we use the AlexNet model as [23], [24]: a CNN with 5 convolutional layers and 3 fully connected layers, requiring 61M weights and 724M multiply-and-accumulates per image. For CIFAR10, we use the VGG16 model [13], [25], which has a deeper architecture consisting of 13 convolutional layers and 3 fully connected layers. It totally requires 138M weights and 15.5G multiply-and-accumulates to process one input image. The two models are often adopted by AI applications [4], [6].

3) HYPERPARAMETERS

The default settings of the FedAvg algorithm, which have been proved to work reasonably well [3], are not optimized in our experiments. For local training, we choose static learning rates of 0.01 for MNIST, 0.1 for CIFAR10, and 0.9 for all momentums. The training data is distributed evenly among the participating clients. In the SHD case, the FL server has $\frac{1}{10}$ and $\frac{1}{5}$ data for AlexNet and VGG16, respectively.

4) HARDWARE

Though in previous works [1], [7] the Raspberry Pi devices are used as IoT clients in FL, training a DNN model for one time on such single-board computers would actually take too much time (ranging from days to weeks), especially for the complex VGG16 model. This would be very challenging and even intolerable for accomplishing all the performance evaluation experiments [4]. Thus, as suggested in [9], we use 3 Intel NUC MiniPCs with Core i5-9300H CPU as clients for AlexNet, and 3 desktop servers with Intel Xeon CPU E5-2678v3 as clients for VGG16. Note that even these CPUs are far less powerful than high-performance GPUs that are commonly used for training DNN models. Meanwhile, the link bandwidth is limited as 1Mbps [7]. It is worth mentioning that the results obtained in this study only provide a reference to demonstrate the effectiveness of FL-PQSU. Actually, when the computation or communication resources of IoT devices are even more constrained, the overheads will be more noticeable and need to be well reduced.

5) BASELINE

In all the experiments, we take the original model training without using any optimization mechanism as the baseline, e.g., when we only adopt the structured pruning mechanism with a pruning ratio of 0.

While the above experimental settings may not be the state-of-the-art, they are sufficient for evaluation, as our intention is to measure the overhead reduction against the baseline but not to achieve the best possible result on the inference tasks.

B. AlexNet TRAINING ON MNIST

In this set of experiments, we prune the network parameters up to 10 levels, from 0 to 90%. When the FL server initially has data samples (SHD), the pruning will be started after the server has trained the initial model training for 30 iterations using its own data.

1) TRAINING ACCELERATION IN ONE ROUND

Fig. 4 shows the speedup on computation and communication (with/without quantization) in a single round, as we vary the

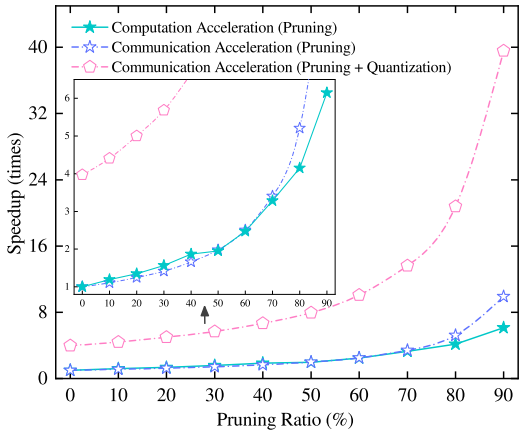


FIGURE 4. Training acceleration by FL-PQSU in one round for AlexNet.

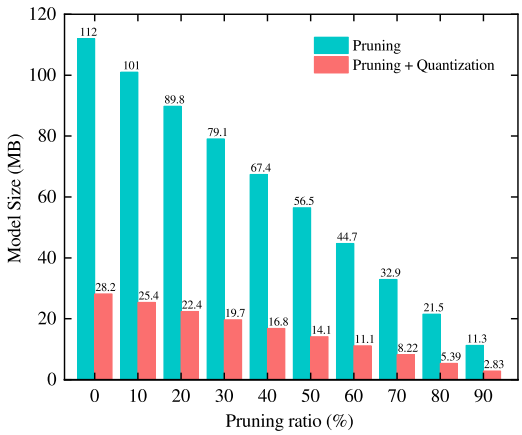


FIGURE 5. Model size reduction by FL-PQSU for AlexNet.

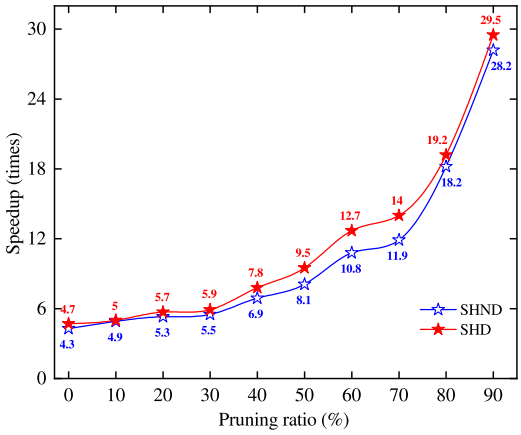


FIGURE 6. Overall training acceleration by FL-PQSU for AlexNet.

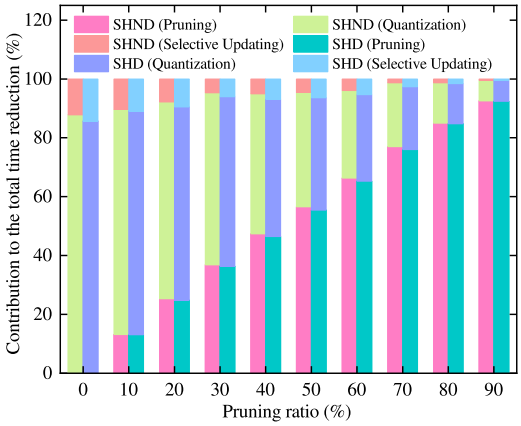


FIGURE 7. Contribution of each strategy in FL-PQSU to acceleration for AlexNet.

pruning ratio. Note that the selective updating strategy is not considered here, as it is adopted only when needed. We see from Fig. 4 that as the pruning ratio increases from 0 to 90%, the time on computation and communication are accelerated by $1 - 6.15\times$ and $1 - 9.91\times$, respectively. Moreover, data quantization from FP32 to INT8 further helps accelerate client-server communication by 4 times, achieving at most $39.58\times$ speedup when the pruning ratio is 90%.

2) MODEL SIZE REDUCTION BY FL-PQSU

Fig. 5 shows the model size under different pruning ratios. As more connections are pruned, the model size decreases from 112 MB to 11.3 MB, and can be further compressed with quantization to 1/4 size on this basis. We can notice that the smallest size is only 2.83 MB, only about 1/40 of the original model size. This reduces not only the cost on communication time but also the requirement on storage space. It is worth noting that in pruning the size reduction is mostly attributed to the pruning to fully connected layers, which are often neglected by conventional pruning studies [13], [14].

3) OVERALL TRAINING ACCELERATION BY FL-PQSU

The results on overall acceleration in model training using all the three strategies are shown in Fig. 6. We can notice an apparent advantage of the SHD-based pruning. In general, the speedup ratio increases from 4.3 $\times$ to 28.2 $\times$ for the SHND case, and from 4.7 $\times$ to 29.5 $\times$ for the SHD case. That's because the initial training at the server make the model more inclined to converge than random initialization, and therefore selective updating can obtain more opportunities to help reduce helpless communications (Fig. 7). We can also observe from Fig. 7 that all the three strategies in FL-PQSU have contributed to the overall time reduction to a certain extent.

4) ACCURACY UNDER DIFFERENT COMPRESSION STRATEGIES

Table 1 shows the inference accuracy under different compression strategies. From the figures and Table 1, we can conclude that FL-PQSU can dramatically speedup FL training for AlexNet, while incurring negligible accuracy loss ($0 - 1.9\%$). Besides, quantization and selective updating (i.e., PQ and PQSU in Table 1) will not further affect the accuracy significantly as compared to pure pruning (i.e, P in Table 1). Further, when all the three strategies (PQSU) are jointly used, FL-PQSU in SHD can achieve an equal or a little higher accuracy as compared to that in SHND under the same pruning ratio.

TABLE 1. Accuracy of AlexNet under different compression strategies. P: Pruning, Q: Quantization, SU: Selective updating.

Pruning Ratio (%)	Accuracy(%)					
	SHND			SHD		
	P	PQ	PQSU	P	PQ	PQSU
0	98.9	98.9	98.7	98.9	99.0	98.8
10	98.8	98.8	98.8	98.9	98.8	98.9
20	98.8	98.8	98.7	98.8	98.8	98.8
30	98.8	98.7	98.7	98.8	99.0	98.7
40	98.8	98.6	98.5	98.5	98.8	98.7
50	98.7	98.6	98.3	98.6	98.5	98.4
60	98.7	98.3	98.3	98.2	98.4	98.3
70	98.1	97.9	97.9	98.0	98.2	98.3
80	97.7	97.7	97.7	98.0	97.9	98.0
90	97.4	97.3	97.3	97.0	97.0	97.5

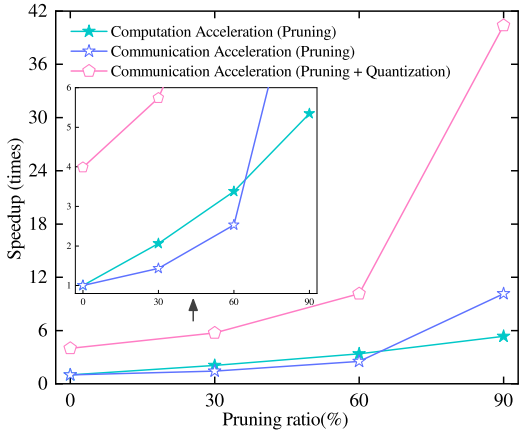


FIGURE 8. Training acceleration by FL-PQSU in one round for VGG16.

C. VGG16 TRAINING ON CIFAR10

To study how the proposed framework performs when the model is deeper and more complex, we evaluate FL-PQSU on VGG16. Different from previous experiments, we only set 4 pruning levels between 0 and 90%, and pre-train the model at the server for 50 iterations in the SHD case.

1) TRAINING ACCELERATION IN ONE ROUND

As shown in Fig. 8, with the increment of pruning ratio, the time on computation and communication are accelerated by $1-5.35\times$ and $1-10.16\times$, respectively. The results are different from those in Fig. 4, as these two DNN models have totally different layer architectures and proportions. Furthermore, the maximal speedup of $40.38\times$ can be reached with data quantization.

2) MODEL SIZE REDUCTION BY FL-PQSU

As shown in Fig. 9, we can prune the model from 128 MB to 12.6 MB. By INT8 quantization, the smallest model is only 3.17 MB.

3) OVERALL TRAINING ACCELERATION BY FL-PQSU

From Fig. 10, we can find that the differences on speedup between SHD and SHND are not as much as those of AlexNet (Fig. 6). That's because VGG16 has a slower convergence process, in which most training iterations are important. Accordingly, the selective updating strategy has little

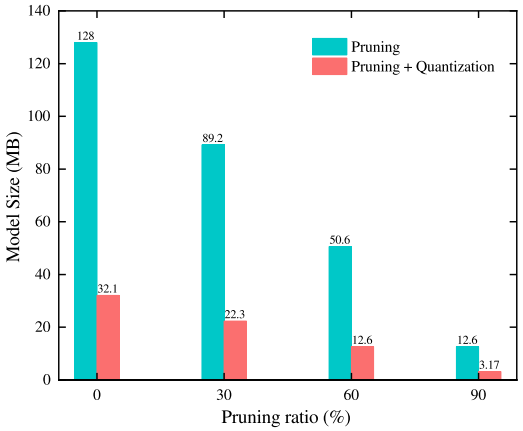


FIGURE 9. Model size reduction by FL-PQSU for VGG16.

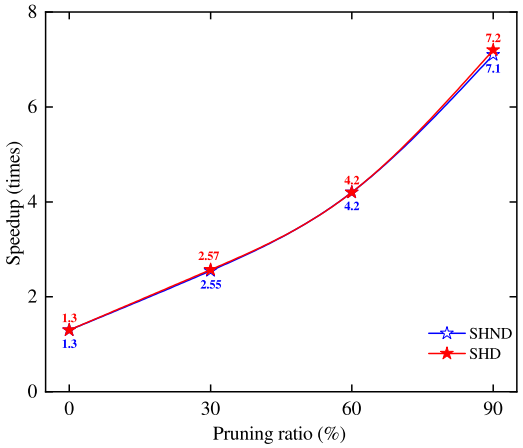


FIGURE 10. Overall training acceleration by FL-PQSU for VGG16.

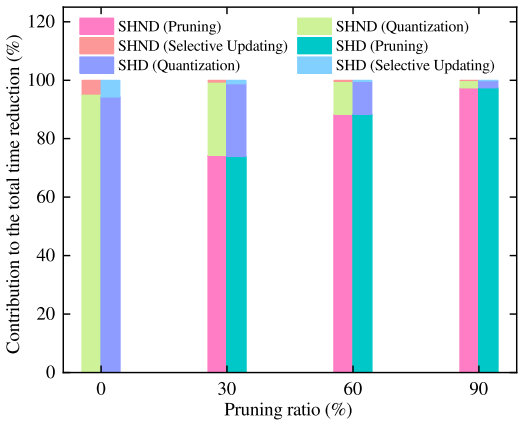


FIGURE 11. Contribution of each strategy in FL-PQSU to training acceleration for VGG16.

contribution to training acceleration, e.g., only about 0.5% when the pruning ratio is 90% (Fig. 11). From Fig. 11, we can find that pruning contributes the most to training time reduction among the three strategies, and selective updating has the least contribution.

4) ACCURACY UNDER DIFFERENT COMPRESSION STRATEGIES

Compared to the results of AlexNet in Table 1, we can make three key observations from Table 2. Firstly, when the pruning

TABLE 2. Accuracy of VGG16 under different compression strategies. P: Pruning, Q: Quantization, SU: Selective updating.

Pruning Ratio (%)	Accuracy(%)					
	SHND			SHD		
	P	PQ	PQSU	P	PQ	PQSU
0	89.9	89.3	90.1	90.9	90.8	90.6
30	91.3	90.3	87.8	92.0	91.2	88.9
60	88.6	88.2	85.8	91.4	90.4	88.6
90	84.9	83.3	81.7	85.8	85.0	84.7

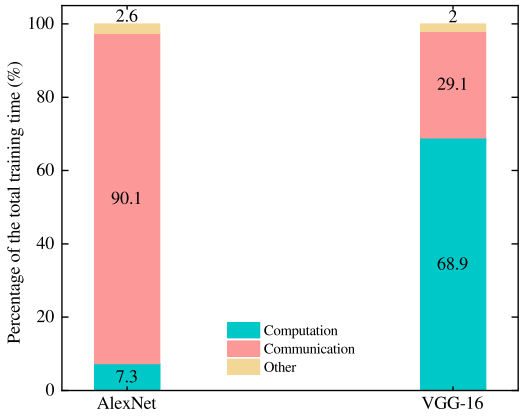


FIGURE 12. Comparisons on the time cost between AlexNet and VGG16.

ratio is relatively low (e.g., $\leq$ 30%), the pruning can even help to improve the accuracy. We hypothesize that the risk of model overfitting could be partially mitigated by pruning redundant parameters. Secondly, when the pruning ratio is relatively high (e.g., $\geq$ 90%), the accuracy loss is as high as 5% $-$ 8.2%, and can't be simply neglected. The results imply that a proper ratio for pruning should be carefully selected beforehand through empirical studies (as there is no strict mathematical statement about relationship between the pruning ratio and the prediction accuracy for DNN by now). Thirdly, different from those in Table 1, the results in Table 2 reveal that when we start to prune the model, quantization and selective updating (i.e., PQ and PQSU in Table 2) will further affect its ability of feature extraction, and lead to non-negligible accuracy loss for complex model and dataset. Lastly, the pre-training at the server can potentially help FL-PQSU to improve its prediction capability, resulting in significant improvements on accuracy for SHD over SHND.

D. COMPARISONS BETWEEN AlexNet AND VGG16

As stated above, we obtain different performance results on speedup and accuracy for AlexNet and VGG16. To investigate the underlying reason, we measure the time consumption on different operations in the FL training process for these models, and compare the results in Fig. 12. For AlexNet, it takes the most of training time (i.e., 90.1%) for client-server communication, which can be well optimized through the three compression strategies by reducing model size, transfered bits, and update times. Comparatively, VGG16 is a more complex model that needs more resource and time (i.e., 68.9%) for convolutional computation. We have to rely more on pruning to speedup the training, which may inevitably hurt the prediction accuracy when the pruning ratio is relatively high.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a new framework FL-PQSU to accelerate the model training in FL on resource-constrained IoT devices. It operates by three key optimizations, including pruning the unimportant connections, quantizing the transfered weights, and exclude the helpless updates. Through extensive experiments on real devices, we have shown that FL-PQSU can achieve the convergence and preserve the benefit of FL, while significantly reduce the costs of computation, storage, and communication. Altogether, FL-PQSU can accelerate the training time of AlexNet for MNIST by 4.3$-$29.5$\times$ and of VGG16 for CIFAR10 by 1.3$-$7.2$\times$, while incurring acceptable accuracy loss or even accuracy gain. With FL-PQSU, we can not only accelerate the training and deployment of DNN models for big data analytics, but also avoid long-time computation to promote active participations of IoT devices.

For the future work, we aim at testing FL-PQSU with typical IoT devices and more models/datasets [4], improving the performance of FL-PQSU on non-IID data [7], and investigating other compression mechanisms that can be applied to FL [2].

REFERENCES

[1] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in IoT," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5986–5994, Jul. 2020.

[2] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 3rd Quart., 2020.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[5] Z. Tao and Q. Li, "ESGD: Communication efficient distributed deep learning on the edge," in *Proc. USENIX Workshop Hot Topics Edge Comput.*, 2018, pp. 1–6.

[6] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[7] Y. Jiang, S. Wang, V. Valls, B. Jun Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," 2019, *arXiv:1909.12326*. [Online]. Available: http://arxiv.org/abs/1909.12326

[8] J. Koneäný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*. [Online]. Available: http://arxiv.org/abs/1610.05492

[9] L. Wang, W. Wang, and B. Li, "CMFL: Mitigating communication overhead for federated learning," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, Dec. 2019, pp. 954–964.

[10] S. Caldas, J. Koneäny, H. Brendan McMahan, and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," 2018, *arXiv:1812.07210*. [Online]. Available: http://arxiv.org/abs/1812.07210

[11] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. Huat Tan, Z. Li, D. Fan, X. Qian, X. Lin, K. Ma, and Y. Wang, "Non-structured DNN weight pruning—Is it beneficial in any platform?" 2019, *arXiv:1907.02124*. [Online]. Available: http://arxiv.org/abs/1907.02124

[12] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. 5th Int. Conf. Learn. Represent.*, 2017, pp. 1–13.
[14] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2736–2744.
[15] P. Singh, V. Kumar Verma, P. Rai, and V. P. Namboodiri, "Play and prune: Adaptive filter pruning for deep model compression," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 3460–3466.
[16] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2020, vol. 34, no. 7, pp. 12273–12280.
[17] S. Lym, E. Choukse, S. Zangeneh, W. Wen, S. Sanghavi, and M. Erez, "PruneTrain: Fast neural network training by dynamic sparse model reconfiguration," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2019, pp. 1–13.
[18] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.
[19] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016, *arXiv:1605.04711*. [Online]. Available: http://arxiv.org/abs/1605.04711
[20] Y. Dong, R. Ni, J. Li, Y. Chen, H. Su, and J. Zhu, "Stochastic quantization for learning accurate low-bit deep neural networks," *Int. J. Comput. Vis.*, vol. 127, nos. 11–12, pp. 1629–1642, Dec. 2019.
[21] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.
[22] A. Briot, P. Viswanath, and S. Yogamani, "Analysis of efficient CNN design techniques for semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2018, pp. 663–672.
[23] S. Banerjee and S. Chakraborty, "Deepsub: A novel subset selection framework for training deep learning architectures," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 1615–1619.
[24] Y. Shima, Y. Nakashima, and M. Yasuda, "Handwritten digits recognition by using CNN alex-net pre-trained for large-scale object image dataset," in *Proc. 3rd Int. Conf. Multimedia Syst. Signal Process.*, 2018, pp. 36–40.
[25] S.-K. Chao, Z. Wang, Y. Xing, and G. Cheng, "Directional pruning of deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1–13.

WENYUAN XU received the B.S. degree from the School of Software, Zhengzhou University, in 2019. He is currently pursuing the degree with the School of Computer and Information Technology, Beijing Jiaotong University. His main current research interests include federated learning and the Internet of Things.

WEIWEI FANG received the B.S. degree from the Hefei University of Technology, Hefei, China, in 2003, and the Ph.D. degree from Beihang University, Beijing, China, in 2010. He is currently an Associate Professor with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing. He has published over 60 papers in journals, international conferences/workshops. His research interests include edge computing, cloud computing, and the Internet of Things.

YI DING received the Ph.D. degree from Beihang University, Beijing, China, in 2014. He is currently an Assistant Professor with the School of Information, Beijing Wuzi University. His current research interests include edge computing and blockchain systems.

MEIXIA ZOU received the B.S. degree from the School of Software, Zhengzhou University, in 2019. She is currently pursuing the degree with the School of Computer and Information Technology, Beijing Jiaotong University. Her main current research interests include DNN compresssion and edge computing.

NAIXUE XIONG (Senior Member, IEEE) received the Ph.D. degree in sensor system engineering from Wuhan University, and the Ph.D. degree in dependable sensor networks from the Japan Advanced Institute of Science and Technology. He was with Northeastern State University, Georgia State University, Wentworth Technology Institution, and Colorado Technical University (Full Professor about five years) about ten years. He is currently a Professor with the College of Intelligence and Computing, Tianjin University. He has published over 300 international journal articles and over 100 international conference papers. Some of his works were published in the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the IEEE or ACM TRANSACTIONS, ACM Sigcomm Workshop, IEEE INFOCOM, ICDCS, and IPDPS. His research interests include cloud computing, security and dependability, parallel and distributed computing, networks, and optimization theory. He is a Senior Member of the IEEE Computer Society. He has been the General Chair, the Program Chair, the Publicity Chair, a PC Member, and an OC Member of over 100 international conferences. He is also the Chair of Trusted Cloud Computing Task Force, the IEEE Computational Intelligence Society (CIS), and the Industry System Applications Technical Committee. He received the Best Paper Award in the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08) and the Best Student Paper Award in the 28th North American Fuzzy Information Processing Society Annual Conference (NAFIPS2009). He is a Reviewer of about 100 international journals, including the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the IEEE SMC (Park: A/B/C), the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE TRANSACTIONS ON MOBILE COMPUTING, and the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He is serving as the Editor-in-Chief and an Associate Editor or an Editor Member for over ten international journals, including an Associate Editor for the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS: SYSTEMS, *Information Science*, the Editor-in-Chief for the JOURNAL OF INTERNET TECHNOLOGY (JIT) and the JOURNAL OF PARALLEL AND CLOUD COMPUTING (PCC), and a Guest Editor for over ten international journals, including *Sensors Journal*, *WINET*, and *MONET*.