Automatic Structured Pruning for Efficient Architecture in Federated Learning

Thai Vu Nguyen, Long Bao Le, Anderson Avila [1]

[1]INRS-EMT, University of Quebec

Abstract

In Federated Learning (FL), training is conducted on client devices, typically with limited computational resources and storage capacity. To address these constraints, we propose an automatic pruning scheme tailored for FL systems. Our solution improves computation efficiency on client devices, while minimizing communication costs. One of the challenges of tuning pruning hyper-parameters in FL systems is the restricted access to local data. Thus, we introduce an automatic pruning paradigm that dynamically determines pruning boundaries. Additionally, we utilized a structured pruning algorithm optimized for mobile devices that lack hardware support for sparse computations. Experimental results demonstrate the effectiveness of our approach, achieving accuracy comparable to existing methods. Our method notably reduces the number of parameters by 89% and FLOPS by 90%, with minimal impact on the accuracy of the FEMNIST and CelebFaces datasets. Furthermore, our pruning method decreases communication overhead by up to 5x and halves inference time when deployed on Android devices. [1]

1 Introduction

Recently, we have witnessed an increasing interest in shifting cloud computing to edge computing (Li et al. 2021c). In fact, bringing computation to the edge of computer networks can reduce latency, benefiting real-time applications, such as autonomous driving (Jin et al. 2024). On the other hand, the data generated, at an unprecedented rate, from billions of edge devices can be used for training and improving AI models (Jia et al. 2024). In such scenarios, Federated Learning (FL) has emerged as a promising alternative to process such an extensive amount of data while preserving users' privacy (McMahan et al. 2017). This is achieved by decentralizing the training process of machine learning (ML) models and keeping personal user data on clients' devices. Thus, training is performed locally and only the model parameters are sent to the central server. The final global model is attained via aggregation of the parameters from all clients.

In the FL settings, where the primary training process occurs on devices with limited computational and storage resources, there is growing interest in developing strategies to decrease the model's footprint. Several studies, for instance, have proposed neural network pruning to reduce model size. The frameworks proposed in (Li et al. 2021a) and (Isik et al. 2023) apply the lottery ticket hypothesis to the FL settings. The work presented in (Qiu et al. 2022) selects the top-k weights of local models to be sent back to the central server for aggregation. In (Bibikar et al. 2022), authors propose a sparse training approach tailored specifically for the FL settings. A crucial limitation of such approaches is that they require pruning to be performed on the local devices. This results in higher computational costs when compared to the normal local training. Additionally, the differences in local models lead to different pruning masks, complicating the aggregation of these masks into the unified global model. To overcome these problems, we propose a pruning procedure that is executed on the central server rather than on individual local devices.

It is important to note that the aforementioned works rely on unstructured pruning, which removes individual weight elements and results in a sparse network with many zero values. Performing fast computations on sparse matrices requires support from specialized libraries, such as cuSPARSE, or hardware (e.g., NVIDIA Ampere GPU) and clients' devices participating in FL training often lack such requirements. Additionally, maintaining sparse data structures requires extra storage for information like compressed sparse rows or binary masks. Thus, inspired by the structured pruning technique presented in (Li et al. 2017), which prunes entire filters rather than individual weights, this paper proposes pruning filters in convolutional networks, maintaining the dense structure of the global model while ensuring compatibility with the simple computational capabilities of client devices.

While centralized learning allows access to the training data and therefore proper tuning of pruning hyper-parameters, such as the sparsity fraction and the number of pruned layers, decentralized learning presents the inability to examine the whole training data directly during training. This makes it difficult to choose a suitable model architecture. For example, in the FL settings, we cannot perform a grid search to determine the optimal number of filters in a convolutional neural network (CNN). To address this challenge, we propose an automatic pruning algorithm, which dynamically defines the pruning boundaries and removes re-

[1]The code implementation is available online: https://github.com/NguyenThaiVu/prune_fl_project.

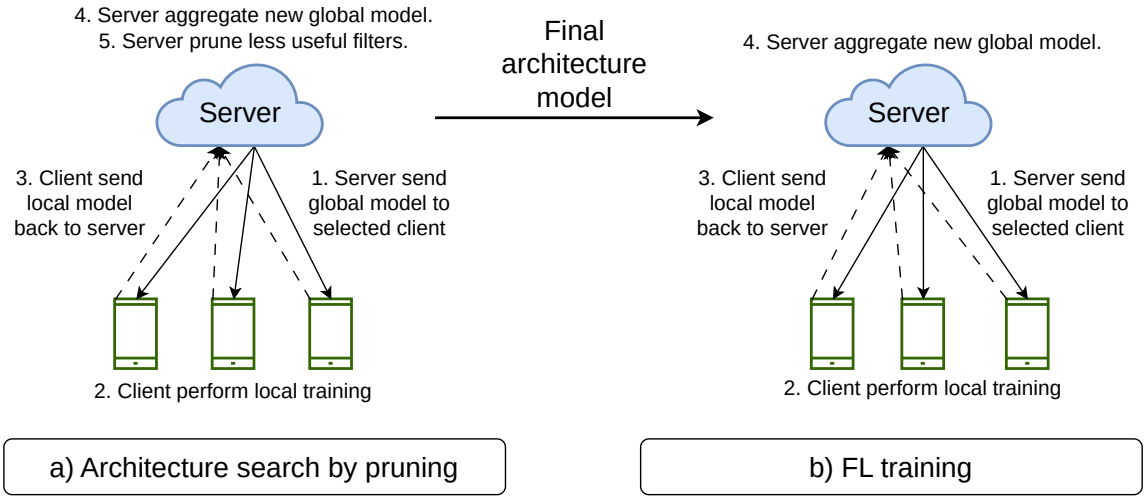

Figure 1: An overview of our pruning scheme in Federated Learning system.

dundant filters without disrupting the FL procedure.

The proposed framework for pruning CNNs in the FL settings is illustrated in Figure 1. The FL training procedure is divided into two sub-stages. First, we perform automatic pruning to identify the well-suited architecture, as depicted in Figure 1-a. This phase involves dynamically eliminating unnecessary filters to reduce the model's size without yet considering the accuracy of the global model. After obtaining the final global model architecture, we proceed with the standard FL training phase (Figure 1-b). The second stage is dedicated to training the final global model to achieve the desired accuracy for the specific task.

Below, we summarize the four main contributions of our work:

- We designed a pruning scheme for FL that significantly reduces redundant parameters/FLOPS in the global model while maintaining the desired accuracy. Additionally, the pruning procedure runs on the central server, reducing computational burdens for local clients.
- We proposed a structured pruning algorithm that automatically determines the pruning boundary and preserves a dense network structure, ensuring fast computation on basic client hardware.
- Extensive experiments show that our pruning algorithm is effective across various architectures, including CNNs, ResNet, and Inception. Additionally, the pruned models significantly reduce inference time on real-world Android devices.
- We also present supplemental studies showing improvements in communication costs and consistency when training with varying random numbers of selected clients in the FL system.

The rest of the paper is organized as follows. Section 2 presents the related work in network pruning, besides summarizing the differences between our paper and existing work. Section 3 introduces our proposed method to prune convolutional neural networks in the FL settings. In Section 4, we conduct extensive experiments to validate the superior performance of our method, followed by the concluding remarks in Section 5.

2 Related Works

2.1 Network Pruning in Federated Learning

Pruning techniques are widely used to reduce the size of neural network models. These approaches are crucial to facilitate the execution of such models on resource-constrained devices (e.g., mobile and IoT gadgets). Pruning methods can be categorized into two main types: unstructured pruning (Han et al. 2015) and structured pruning (Li et al. 2017). Unstructured pruning involves removing individual weights in the network, resulting in a sparse weight matrix. In contrast, structured pruning implies the elimination of entire components of the network, such as layers or filters, while maintaining a dense network architecture.

Due to the decentralized nature of FL, applying unstructured pruning on each client may result in differently pruned networks, leading to inconsistencies during the aggregation of global models. Some studies have leveraged these varying pruning masks for personalized federated learning (Li et al. 2021b,a; Dai et al. 2022). Other researchers (Bibikar et al. 2022; Isik et al. 2023; Qiu et al. 2022; Babakniya et al. 2023; Jiang et al. 2023b; Jiang and Borcea 2023; Huang et al. 2023) have proposed methods to aggregate different pruning masks into the global sparse model. However, the resulting global model still contains a sparse weight matrix, which can lead to slower inference times on limited client devices.

On the other hand, structured pruning emerges as an ideal choice for deployment on edge devices in the FL system. This method preserves dense network architectures, facilitating straightforward deployment on simple hardware without the need for specialized sparse matrix support. In (Vahidian, Morafah, and Lin 2021; Wu, Yao, and Wang 2020), for

instance, the pruning process occurs at the edge of the network, increasing the computational burdens of less powerful mobile/IoT devices. In contrast, the authors in (Xu et al. 2021) propose to perform pruning on a central server. Their method prunes the model only once, during the initialization of the global model, excluding the pruning procedure from the training process. The method presented in (Zhang et al. 2022) assumes that the training dataset is located on the central server, which contradicts the decentralized nature of FL The FedPara method (Hyeon-Woo, Ye-Bin, and Oh 2022) employs the low-rank Hadamard product to reduce the number of parameters. Nevertheless, the low-rank matrix approximation still occurs on client devices, increasing local computation.

Unlike previous work, our method performs structured pruning on a powerful central server, offering several advantages. First, structured pruning maintains the dense network structure, facilitating straightforward hardware implementation. Second, by conducting the pruning procedure on the central server, we eliminate additional computational burdens on client devices. Finally, because pruning is performed centrally, our method does not require an aggregation mechanism to combine different pruned client models into a global model. Instead, we utilize the standard FedAvg algorithm for model aggregation.

2.2 Pruning Hyper-parameter

Previous works primarily rely on the pre-defined pruning hyper-parameters such as accuracy threshold, sparsity ratio, or pruning rate to control the pruning procedure. The method (Li et al. 2021a) employs a predefined pruning rate to determine the number of pruned parameters per client. However, selecting an appropriate pruning rate is challenging, as a rate effective for current devices may not be suitable for others. PruneFL (Jiang et al. 2023a) relies on powerful and trusted clients to find suitable architecture. However, finding powerful clients is challenging in a heterogeneous FL system. Other methods (Bibikar et al. 2022; Qiu et al. 2022; Babakniya et al. 2023) utilize pre-defined sparsity density, or sparsity ratio to manage the pruning procedure. The FedDUAP (Zhang et al. 2022) adopts server data to halt the pruning method. Additionally, methods (Li et al. 2021b; Hyeon-Woo, Ye-Bin, and Oh 2022) rely on predefined lower rank values or hard thresholds to reduce the number of parameters. However, in the decentralized FL, selecting appropriate hyper-parameters poses a challenge due to the absence of central training data, which are critical for determining optimal hyper-parameters.

Unlike previous approaches, our automatic pruning algorithm simplifies the hyper-parameter selection process by removing the need to specify sparsity ratios or pruning rates. Instead, it automatically identifies and removes redundant filters within the model. Additionally, by not assigning specific sparsity ratios to each client, we prevent pruning discrepancies that can occur when a pre-defined ratio is suitable for one client but inappropriate for others. This approach ensures consistent performance of our pruning algorithm across heterogeneous clients.

3 Structured Pruning ConvNets for Federated Learning

In this section, we present the proposed two stages FL training procedure. We first describe the automatic pruning filter algorithm used in the first stage. We then outline the procedure applied in the second stage for pruning network architecture in FL training. We end by showing that the remained filters still retain statistical properties of good weight initialization.

3.1 Automatic Pruning Algorithm

There are two approaches for determining the number of layers to prune: uniform pruning and automatic pruning (Liu et al. 2019). In uniform pruning, the same number of weights/filters is removed at each epoch throughout the training process. The uniform pruning works well in centralized learning, where the entire training dataset is fully accessible, facilitating the selection of optimal architecture based on validation performance criteria. However, in the context of FL, automatic pruning becomes an intelligent strategy due to the challenge of selecting optimal pruning hyper-parameters, without entirely access to the training data.

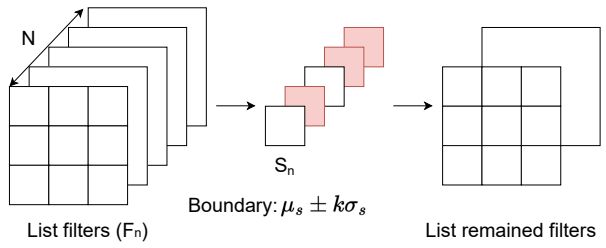

Figure 2: An overview of our structured pruning algorithm, which assesses convolution filters based on the sum of their absolute values S_n. Filters that fall outside the boundary $\mu_s \pm k\sigma_s$ (where μ_s is the mean and σ_s is the standard deviation of S_n) are pruned and highlighted in red.

To effectively perform automatic pruning, we proposed Algorithm 1. This algorithm not only automatically eliminates redundant filters, but also retains the dense structure of the model architecture. The illustration of the pruning algorithm is described in Figure 2. Although Algorithm 1 is specifically described for 2D convolutional filters, it can be extended to other types of convolutional filters, such as 1D and 3D filters. For 1D filters, the algorithm would sum the absolute values of elements along a single dimension. For 3D filters, the algorithm would sum the absolute values across three dimensions. In both cases, the computation of the mean, standard deviation, and pruning criteria should be appropriately adapted to account for the dimension of the filters.

The utilization of the mean (μ_s) and standard deviation (σ_s) of the sum absolute values (S_n) across all filters to establish a pruning boundary offers a statistical intuition for assessing the influence of filters on the prediction. Filters exhibiting lower S_n values contribute minimally to the out-

Algorithm 1: Pruning filter Algorithm

Input:

- $W \in \mathbb{R}^{N \times K \times K}$: initialized weight matrix
- N: number of 2D filters
- $K \times K$: filter dimension
- k: constant defining pruning boundary

Output:

- $W' \in \mathbb{R}^{N' \times K \times K}$: remained weight matrix

Algorithm:

Initialize S as a vector of length N
for $n = 1$ to N **do**
  $S_n = \sum_{i=1}^{K} \sum_{j=1}^{K} |W_{nij}|$
end for
Mean $\mu_s = \frac{1}{N} \sum_{n=1}^{N} S_n$
Standard deviation $\sigma_s = \sqrt{\frac{1}{N} \sum_{n=1}^{N} (S_n - \mu_s)^2}$
Lower bound $= \mu_s - k\sigma_s$
Upper bound $= \mu_s + k\sigma_s$
Initialize an empty list W' to store remained filters
for $n = 1$ to N **do**
  if $\mu_s - k\sigma_s \leq S_n \leq \mu_s + k\sigma_s$ **then**
    Add W_n to W'
  end if
end for
Return W'

put, suggesting limited contribution. Conversely, exceptionally high S_n values may introduce excessive noise, potentially disrupting the output. By incorporating a scaling factor k into the boundary definition, the algorithm allows for flexible adjustment between aggressive pruning (lower k) and performance retention (higher k).

For straightforward CNN architectures (AlexNet or VGG network), layer-by-layer pruning is easily applicable, as discussed in Algorithm 1. However, in more complicated architectures, such as residual blocks (He et al. 2016) or inception modules (Szegedy et al. 2015), pruning filters become more complex. Firstly, in a residual block (Figure 3.a), pruning the first convolutional layer is permissible because it does not alter the output shape. However, pruning the second convolutional layer is not advisable, as it directly affects the output shape of the residual block, potentially disrupting the entire network. Secondly, in inception modules (Figure 3.b), all convolutional layers can be pruned independently. This flexibility stems from the inception module's design, which employs concatenation operations at the end rather than requiring consistent shapes through element-wise operations. As a result, pruning filters within an inception block can be done individually without affecting other parts of the network.

3.2 Decentralized Training Procedure

The overall procedure of the proposed pruning method in the FL system is illustrated in Figure 1, which includes two main stages: architecture search by pruning and FL training.

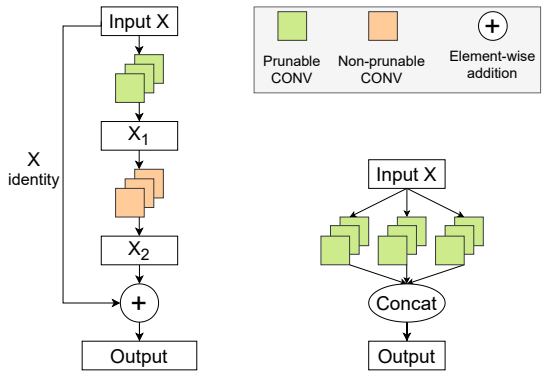

Figure 3: An overview of pruning procedure for complex designed convolutional architecture. Prunable Conv indicates layers can be pruned by Algorithm 1. Non-prunable Conv is layers, which have a fixed number of filters and can not be pruned.

Stage 1: Architecture search by pruning. The automatic pruning Algorithm 1 is applied to the global model at each round of FL training. The detailed procedure for the architecture search stage is depicted in Figure 1-a, and is described as follows:

1. The server sends a global model to the selected subset of clients.
2. Clients receive the global model and perform local training on private datasets.
3. All clients send back the trained local model to the server.
4. The server aggregates all local models into the new global model.
5. The server prunes the new global model using the automatic pruning Algorithm 1.

In the above process, steps 1-5 are repeated until discovering a suitable architecture. We adopt a simple early stopping schedule to stop the architecture search stage: after c rounds (c=3 in the experiment), if the number of parameters in the global model does not decrease, we halt the architecture search stage. That showcases another advantage of automatic pruning: there is no need to explicitly specify the number of pruning rounds, as the determination is made automatically during the training process.

In step 4, we utilize the standard FedAvg (McMahan et al. 2017) on each round to aggregate all trained local models into a new global model. Since clients do not prune their local models, all of which share the same architecture, no additional aggregation scheme is necessary. Moreover, because pruning occurs on the server side, it mitigates potential issues for less powerful client devices.

Stage 2: Federated Learning training. After the first architecture search stage, we obtain the final global model architecture. Subsequently, we proceed with the standard Federated Learning training stage utilizing the selected architecture. The primary objective of this second stage is to fully

(a) Performance on FEMNIST dataset

Architecture	# Conv	Number params	Pruned % params	FLOPS	Pruned % FLOPS	Accuracy
Conv	2	56 126		7.5×10^6		74.72
Conv-pruned	2	16 683	70%	2.2×10^6	70%	74.22
ResNet	7	33 214		5.2×10^5		79.29
ResNet-pruned	7	5 668	82%	0.9×10^5	82%	74.55
Inception	9	246 236		31×10^6		80.19
Inception-pruned	9	26 744	89%	3×10^6	90%	78.81

(b) Performance on CelebFaces dataset.

Architecture	# Conv	Number params	Pruned % params	FLOPS	Pruned % FLOPS	Accuracy
Conv	4	80 930		152×10^6		95.87
Conv-pruned	4	10 998	86%	29×10^6	80%	92.59
ResNet	7	30 626		5.2×10^6		89.50
ResNet-pruned	7	6 791	77%	1.6×10^6	69%	85.83
Inception	9	243 752		324×10^6		87.78
Inception-pruned	9	127 080	47%	171×10^6	47%	86.23

Table 1: The results of our pruning method on the FEMNIST and CelebFaces datasets across Convolution, ResNet, and Inception architecture. We report the highest test accuracy achieved.

optimize the model's performance for the specific task we are working on.

4 Experiments

4.1 Experimental Setup

We evaluate the performance of our pruning method on three convolutional neural network architectures: Vanilla Convolution, ResNet, and Inception. The filter size is 5x5, and the number of filters for each model is detailed in Table 1. All models were trained from scratch, using cross entropy as loss function and Adam as optimizer. To simulate real-world conditions as closely as possible, we use the standard LEAF benchmark (Caldas et al. 2018) with the FEMNIST and CelebFaces datasets. The FEMNIST dataset includes more than 800,000 images with size of 28x28 pixels and the task is a digit classification problem with 62 output categories. The CelebFaces dataset includes RGB images with 82x82x3 pixels, with each sample containing a human face to be predicted as male or female by the proposed model. On client devices, we limit the local training to 5 epochs with batch size equal 32 due to the constrained computing power of clients, particularly mobile and IoT devices. Regarding the FL system, we randomly choose 10% clients in each round, with a total of 500 rounds considered in our experiments.

4.2 Baseline Results

We will begin by examining the impact of the pruned model in the Federated Learning (FL) system. The primary objective of this comparison is to highlight the ability of our method to achieve maintain acceptable accuracy with minimal storage cost (number of parameter) computation cost (FLOPS). Table 1 describes the number of parameters and FLOPS reduced after applying pruning. Results are attained from the FEMNIST and CelebFaces datasets. In all cases, our method significantly reduced the number of parameters/FLOPS and the accuracy is maintained to acceptable levels. For instance, in the case of the Inception architecture on the FEMNIST dataset, the number of parameters and FLOPS are reduced by 90% while the accuracy is reduced by only 2%. Despite the complexity of the CelebFaces dataset (200,000 RGB images), our pruning method still achieves promising results. Notably, with the Conv architecture, the number of parameters is reduced by 86% but the accuracy is only reduced by about 3%.

Table 2: The performance comparison between our model and other pruning methods on the FEMNIST dataset.

Method	Prune Type	Accuracy
LotteryFL	unstructured	58.69
FedPM	unstructured	61.29
FedPara	structured	65.52
PruneFL	unstructured	57.09
FlashFL	unstructured	65.39
FeDST	unstructured	52.34
FeDST+FedProx	unstructured	52.94
Our model	structured	**74.22**

In addition, we also compared the accuracy of our pruned model with other methods on the FEMNIST dataset, as shown in Table 2. To ensure comparison closely follows the original papers, we used their publicly repository. The results show that our method achieves 74.22% accuracy, which outperform other methods. One explanation for this result is that our pruning method preserves the dense structure of the model, which facilitates the global model aggregation on the

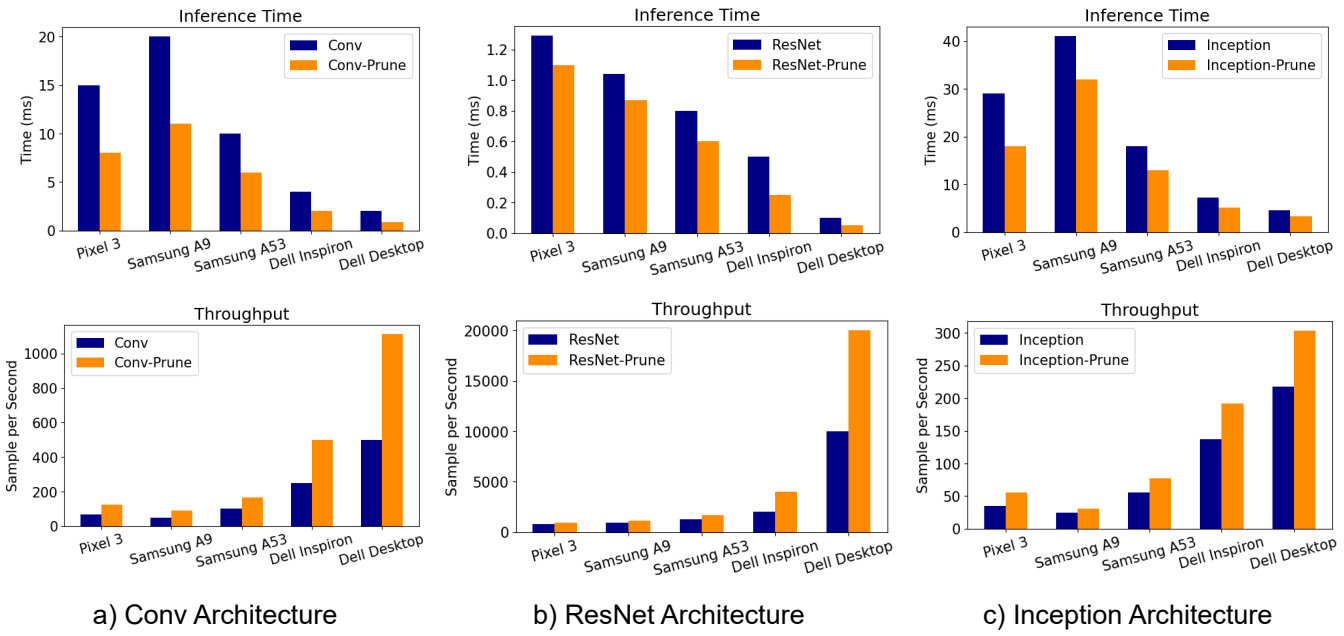

Figure 4: Evaluation of inference time (milliseconds) and throughput (samples per second) across five devices for various model architectures.

Table 3: The hardware information of several testing devices.

Devices	Chip	CPU cores	RAM
Google Pixel 3	Snapdragon 845	8	8 GB
Samsung A9	Snapdragon 660	8	6 GB
Samsung A53	Exynos 1280	8	6 GB
Dell Inspiron 15	Intel i7-7700	4	8 GB
Dell Desktop	Intel i7-4790	8	16 GB

FL system.

4.3 Deployment on Real Devices

In this section, we present the results of deploying the pruned model on various real-world devices, such as smartphones, personal laptops, and desktops. On mobile scenarios, the pruned model was integrated into a Java-based mobile application for Android devices such as the Google Pixel 3, Samsung Galaxy A9, and Samsung Galaxy A53. On desktop scenarios, the model was incorporated into a Python-based application using Streamlit, a framework designed for building interactive web applications, which was executed on a Dell Inspiron laptop and a Dell desktop. The reader can refer to Table 3 for detailed information regarding the devices used herein. By embedding the pruned model into these devices, we were able to assess its performance in a simulated practical real-world environment.

Android devices currently offer limited support for well-known deep learning frameworks such as TensorFlow and PyTorch. To handle this limitation, we converted the trained model into the TensorFlow Lite (tflite) format, which is a lightweight model format for mobile environments. For practical application, we utilize a model trained on the CelebFaces dataset to classify images as male or female. We evaluated the effectiveness of the pruning method by examining inference time and throughput. Inference time refers to the duration required to perform a single inference operation on an image, excluding auxiliary tasks such as image loading and data processing. Throughput is defined as the number of samples (images) the model can process per second. This metric is crucial for evaluating the model's ability to manage a high volume of predictions within a specific time frame.

Figure 4 describes the comparison in terms of inference time and throughput between the original and the pruned models across four different devices. The results indicate that the pruned model cuts the inference time by nearly half and doubles the throughput on all tested devices, demonstrating a substantial improvement in efficiency. For instance, in a standard convolutional architecture, the throughput on a Dell Desktop increased from 500 to 1000 samples per second.

When deploying deep learning models into real-world applications, it is crucial to consider several factors beyond inference time. Figure 5 illustrates a comparison between the original and pruned models based on several criteria, including model size, accuracy, and memory usage. The experiments were conducted using the tflite model format on an Ubuntu Dell Desktop. The outcomes demonstrate that in all tested architectures, the proposed pruning method significantly reduced the required disk space when compared to their original counterpart, making our solution more suitable for devices with storage constraints. Additionally, memory

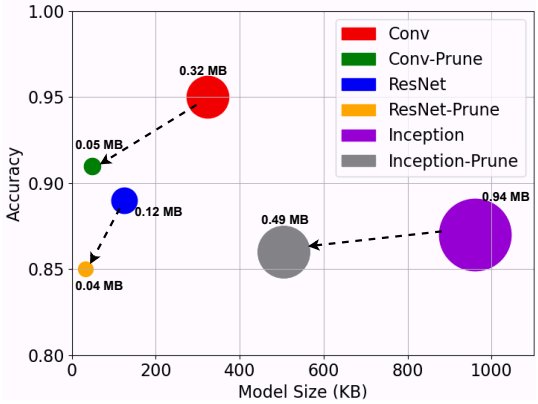

Figure 5: The diagram illustrates the model size and accuracy for various models. The size of each circle and the value adjacent to it represent the memory consumption (RAM) required for running the inference. Directed arrows show the transition from the original model to its pruned counterpart.

consumption (RAM) during the inference phase was significantly lower in the pruned models. This reduction in both model size and memory usage enhances the feasibility of deploying these models on devices with limited resources.

4.4 Communication Cost and Client Drop

In this section, we evaluate communication cost, which is defined as the total amount of data transferred between server and clients during the FL training process. For this experiment, we conducted 100 rounds of FL training and calculated the total communication cost by summing the data transferred in all rounds. The data transferred is determined by the size of the model when sent from the server to the client and vice versa. Specifically, the model size is calculated by multiplying the number of model parameters by the size of each parameter element. In our default setup, we use float32 elements, which are 4 bytes each.

Figure 6 illustrates the communication cost across various convolutional architectures, including Convolution, ResNet, and Inception on the FEMNIST dataset. In all cases, our pruning method significantly reduces the communication cost compared to normal training. Notably, in the experiments involving vanilla Conv (Figure 6.a) and Inception (Figure 6.c), the cost is reduced up to 5 times. This reduction is a result of our model being progressively pruned during the FL training process. Consequently, the required communication cost diminishes and enhances the overall efficiency of the FL process.

In the FL systems, the number of clients participating in each round can vary due to various factors like availability and connectivity. To ensure the robustness of our proposed pruning technique, we conducted a series of experiments. Specifically, we tested the pruning technique with different numbers of selected clients, including 5, 10, and 50 clients per round, which allowed us to evaluate how well the pruning technique performs under different number of client participation. These experiments were performed using the FEMNIST dataset and involved various neural network architectures (Conv, ResNet, and Inception architecture).

The error bar in Figure 7 visualizes the uncertainty associated with the performance of a pruning algorithm in FL system across different numbers of clients. The x-axis represents the different client numbers per round in the FL training process and the y-axis represents the number parameters of the pruned model. On each architecture, the mean value of the retained parameters (represented by the height of each bar) remains approximately the same value when training with different numbers of selected clients. This consistency indicates that our pruning method maintains stable performance regardless of the number of clients participating in each training round. This robustness is particularly noteworthy given the inherent variability in the Federated Learning (FL) process.

4.5 Ablation Study Hyper-parameter k

In this subsection, we conduct an ablation study to deepen our understanding of how the hyper-parameter k affects the performance of pruning Algorithm 1. Specifically, we perform experiments using the convolutional architecture on two datasets FEMNIST and CelebFaces with k values set at 2.0, 2.5, and 3.0 respectively. The Pareto Diagram, shown in Figure 8, illustrates the trade-off between a number of parameters (in thousands) and model accuracy, demonstrating the impact of varying k.

The bar chart, colored in blue and plotted on the left y-axis, represents the number of parameters of the pruned model, illustrating the aggressiveness of the pruning process at each value of k. As k decreases, a greater proportion of the model is pruned, reflecting a more aggressive reduction in model size. Interestingly, when $k = 3$, the pruned model retains nearly the same number of parameters as the original (non-prune) model. Conversely, the line chart on the right y-axis (color in red) displays the corresponding model accuracy. As expected, a clear trend emerges where higher pruning typically corresponds with lower accuracy levels. This pattern highlights the essential trade-offs that need to be considered when optimizing the hyper-parameter k for pruning.

5 Conclusion

In this paper, we presented a structured pruning method for Federated Learning (FL) systems that significantly reduces the model's parameters, computational costs, and communication overhead without introducing sparsity into the weight matrix. In addition, to tackle the challenge of choosing an optimal model architecture in FL environments, we introduced an automatic pruning algorithm that determines the optimal number of filters to prune. Our method effectively reduces up to 90% of parameters and FLOPS on the FEMNIST dataset and 80% on the CelebFaces dataset, with only a minimal loss in accuracy. Additionally, when deployed on real Android devices, the pruned model cuts inference time by up to 50% and doubles the throughput. Furthermore, the pruned model decreases communication costs up to five times and maintains consistent performance across varying numbers of selected clients during the FL procedure.

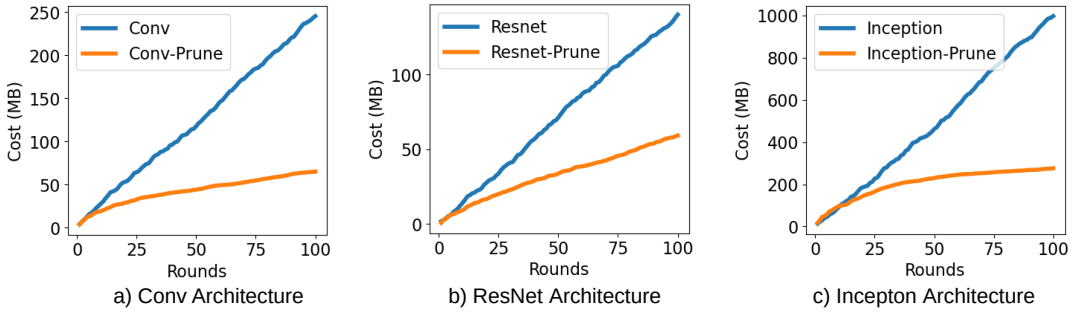

Figure 6: Communication Cost across different convolutional architectures on the FEMNIST dataset.

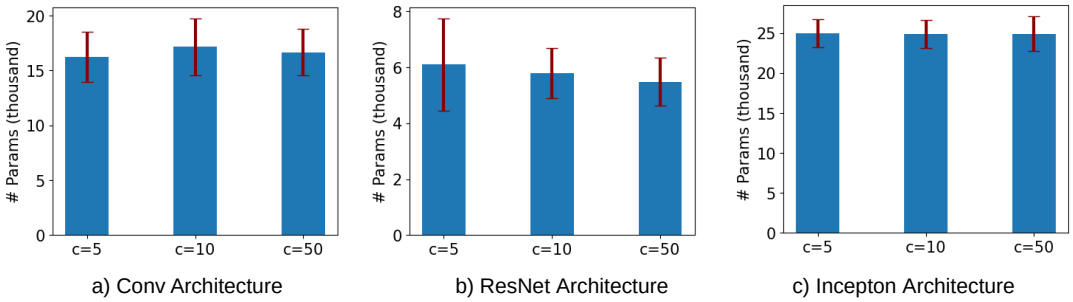

Figure 7: Uncertainty in model pruning across different client configurations on the FL system (c is the number of selected clients). Each error bar represent the mean number of parameters resulting from 10 different experimental runs.

Reference

Babakniya, S.; Kundu, S.; Prakash, S.; Niu, Y.; and Avestimehr, S. 2023. Revisiting Sparsity Hunting in Federated Learning: Why does Sparsity Consensus Matter? *Transactions on Machine Learning Research*.

Bibikar, S.; Vikalo, H.; Wang, Z.; and Chen, X. 2022. Federated Dynamic Sparse Training: Computing Less, Communicating Less, Yet Learning Better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 6080–6088.

Caldas, S.; Duddu, S. M. K.; Wu, P.; Li, T.; Konečnỳ, J.; McMahan, H. B.; Smith, V.; and Talwalkar, A. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* .

Dai, R.; Shen, L.; He, F.; Tian, X.; and Tao, D. 2022. Dispfl: Towards communication-efficient personalized federated learning via decentralized sparse training. In *International conference on machine learning*, 4587–4604. PMLR .

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256. JMLR Workshop and Conference Proceedings .

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034 .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778 .

Huang, H.; Zhang, L.; Sun, C.; Fang, R.; Yuan, X.; and Wu, D. 2023. Distributed Pruning Towards Tiny Neural Networks in Federated Learning. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, 190–201. IEEE .

Hyeon-Woo, N.; Ye-Bin, M.; and Oh, T.-H. 2022. FedPara: Low-rank Hadamard Product for Communication-Efficient Federated Learning. In *International Conference on Learning Representations* .

Isik, B.; Pase, F.; Gunduz, D.; Weissman, T.; and Michele, Z. 2023. Sparse Random Networks for Communication-Efficient Federated Learning. In *The Eleventh International Conference on Learning Representations* .

Jiang, Y.; Wang, S.; Valls, V.; Ko, B. J.; Lee, W.-H.; Leung, K. K.; and Tassiulas, L. 2023a. Model Pruning Enables Efficient Federated Learning on Edge Devices. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12): 10374–10386 .

Jiang, Z.; Xu, Y.; Xu, H.; Wang, Z.; Liu, J.; Chen, Q.; and Qiao, C. 2023b. Computation and communication efficient federated learning with adaptive model pruning. *IEEE Transactions on Mobile Computing*, 23(3): 2003–2021 .

Jiang, X.; and Borcea, C. 2023. Complement sparsification: Low-overhead model pruning for federated learning. In

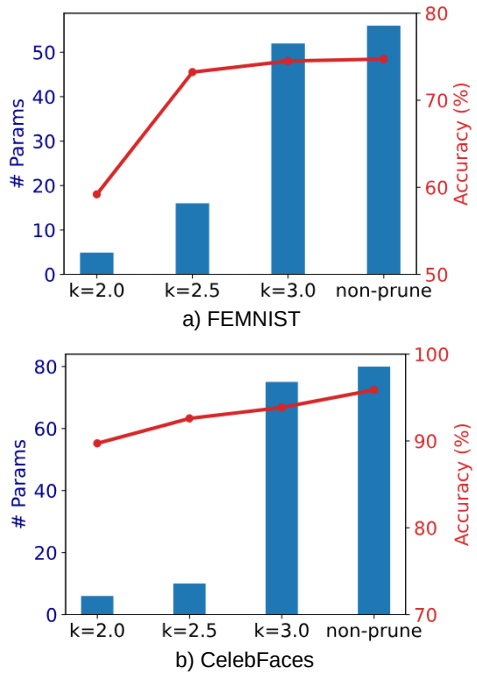

Figure 8: The Pareto Diagram shows the trade-off between pruning aggressive and accuracy retention across different values of hyper-parameter k.

Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, 8087–8095 .

Jia, Y.; Zhang, X.; Beheshti, A.; and Dou, W. 2024. FedLPS: Heterogeneous Federated Learning for Multiple Tasks with Local Parameter Sharing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 12848–12856.

Jin, L.; Tang, M.; Zhang, M.; and Wang, H. 2024. Fractional deep reinforcement learning for age-minimal mobile edge computing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 12947–12955 .

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations* .

Li, A.; Sun, J.; Wang, B.; Duan, L.; Li, S.; Chen, Y.; and Li, H. 2021a. LotteryFL: Empower Edge Intelligence with Personalized and Communication-Efficient Federated Learning. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, 68–79 .

Li, A.; Sun, J.; Zeng, X.; Zhang, M.; Li, H.; and Chen, Y. 2021b. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 42–55 .

Li, L.; Shi, D.; Hou, R.; Li, H.; Pan, M.; and Han, Z. 2021c. To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 1–10. IEEE.

Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; and Darrell, T. 2019. Rethinking the Value of Network Pruning. In *International Conference on Learning Representations* .

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and Arcas, B. A. y. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 1273–1282. PMLR.

Qiu, X.; Fernandez-Marques, J.; Gusmao, P. P.; Gao, Y.; Parcollet, T.; and Lane, N. D. 2022. ZeroFL: Efficient On-Device Training for Federated Learning with Local Sparsity. In *International Conference on Learning Representations* .

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning Efficient Convolutional Networks Through Network Slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* .

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9 .

Vahidian, S.; Morafah, M.; and Lin, B. 2021. Personalized federated learning by structured and unstructured pruning under data heterogeneity. In *IEEE 41st international conference on distributed computing systems workshops (ICDCSW)*, 27–34. IEEE .

Wu, X.; Yao, X.; and Wang, C.-L. 2020. FedSCR: Structure-based communication reduction for federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(7): 1565–1577 .

Xu, W.; Fang, W.; Ding, Y.; Zou, M.; and Xiong, N. 2021. Accelerating federated learning for iot in big data analytics with pruning, quantization and selective updating. *IEEE Access*, 9: 38457–38466 .

Zhang, H.; Liu, J.; Jia, J.; Zhou, Y.; Dai, H.; and Dou, D. 2022. FedDUAP: Federated Learning with Dynamic Update and Adaptive Pruning Using Shared Data on the Server. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 2776–2782. International Joint Conferences on Artificial Intelligence Organization.