NETWORK-AWARE FEDERATED NEURAL ARCHITECTURE SEARCH

by

Göktuğ Öcal

B.S., Control and Automation Engineering, Istanbul Technical University, 2020

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2024

ACKNOWLEDGEMENTS

First of all, I want to express my thanks and gratitude to my dear supervisor Assist. Prof. Atay Özgövde for his endless guidance and encouragement through both the MS education and the thesis study. Working with him is like never-ending brainstorming for life and that made the journey amusing and exciting. His endless sharing of experience lightens up the path while fighting away the darkness. I would like to thank Ali Burak Acar for the acceleration he gave through my thesis study. I want to thank all the instructors of the Department of Computer Engineering of Boğaziçi University.

Obviously, I would thank my lovely family for their sensitivity to education, their dedication that makes everything possible, and their belief in me.

My biggest "thank you" is to my partner, friend, and my wife, Serra Kahraman Öcal. I dedicate this thesis to the person who always pushes me up, believes in me, and makes me smile.

ABSTRACT

NETWORK-AWARE FEDERATED NEURAL ARCHITECTURE SEARCH

Deep Learning (DL) has significantly transformed various aspects of daily life by empowering technological innovations. The cooperation between DL and the Internet of Things (IoT) has further advanced these developments, allowing smart devices to serve as both data sources and endpoints for DL-powered applications. However, the success of DL relies on optimal Deep Neural Network (DNN) architectures, and manually developing such systems requires extensive expertise and time. Automated machine learning methods, such as Neural Architecture Search (NAS), have emerged to automate the search for the best-performing neural architectures. Meanwhile, Federated Learning (FL) addresses data privacy concerns by enabling collaborative model development without exchanging the private data of clients.

This thesis focuses on the collaboration of Federated Learning and Neural Architecture Search, addressing challenges exposed by IoT devices with limited resources and changing network conditions. Contributions include the development of an open-source federated neural network pruning framework with network emulation support, aiming to simulate real-world scenarios and efficiently reduce the size of DNNs in a distributed network. A novel client grouping algorithm is proposed to optimize communication efficiency, considering both varying class distributions and network bandwidth conditions. Moreover, a detailed exploration of the impact of network bandwidth on federated learning, especially communication rounds, provides valuable insights for designing robust federated learning systems. These contributions collectively enhance the understanding of federated neural architecture search where computational efficiency, data privacy, and network conditions show significant roles.

ÖZET

AĞ DUYARLI FEDERE SİNİR MİMARİSİ ARAMASI

Derin Öğrenme, teknolojik yenilikleri şekillendirerek dünyayı farklı yönlerde önemli ölçüde etkiledi. DL ile Nesnelerin İnterneti arasındaki işbirliği, bu gelişmeleri daha da ileri taşıyarak akıllı cihazların DL destekli uygulamalar için hem veri kaynağı hem de uç nokta olarak hizmet etmesine olanak tanıdı. Ancak DL'nin başarısı, optimum Derin Sinir Ağı mimarilerine dayanır ve bu tür sistemleri geliştirmek, kapsamlı uzmanlık ve zaman gerektirir. Sinir Mimarisi Araması gibi otomatik makine öğrenimi yöntemleri, en iyi performansa sahip sinir mimarilerinin aranmasını otomatikleştirmek için geliştirilmiştir. Bu sırada, Federe Öğrenme, cihazların ve kullanıcıların özel verilerinin paylaşılması gerekmeden işbirlikçi model geliştirmeyi mümkün kılarak veri gizliliği endişelerini giderdi.

Bu tez, sınırlı hesaplama kaynaklarına ve değişen ağ koşullarına sahip IoT cihazlarının maruz kaldığı zorlukları ele alan Federe Öğrenme ve Sinir Mimarisi Araması arasındaki işbirliğine odaklanmaktadır. Katkılar arasında, gerçek dünyadaki senaryoları simüle etmeyi ve dağıtılmış bir ağdaki modellerin boyutunu verimli bir şekilde azaltmayı amaçlayan, ağ emülasyon desteğine sahip açık kaynaklı bir federe sinir ağı budama aracının geliştirilmesi yer alıyor. Hem cihazların hassas verilerinin sınıf dağılımlarını hem de ağ bant genişliği koşullarını dikkate alarak iletişim verimliliğini optimize etmek için yeni bir istemci gruplama algoritması önerilmiştir. Ayrıca, ağ bant genişliğinin federe öğrenme, özellikle de iletişim turları üzerindeki etkisinin ayrıntılı bir şekilde araştırılması, verimli federe öğrenme sistemlerinin tasarlanması için faydalı sonuçlar sağlamıştır. Hesaplama verimliliğinin, veri gizliliğinin ve ağ koşullarının önemli rol oynadığı IoT sistemlerinde federe sinir mimarisi araştırması uygulanmasına katkı sağlamıştır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES viii
LIST OF TABLES xi
LIST OF SYMBOLS xii
LIST OF ACRONYMS/ABBREVIATIONS xiii
1. INTRODUCTION 1
2. BACKGROUND 6
2.1. Neural Architecture Search and Neural Network Pruning 6
2.1.1. Neural Architecture Search 6
2.1.2. Neural Network Pruning 9
2.2. Federated Learning 13
2.2.1. Cloud Computing and Data Management 13
2.2.2. Federated Learning 16
2.2.3. Weight Aggregation Algorithms 18
2.3. IID and Non-IID Data Distributions 21
2.3.1. Dirichlet Distribution 22
2.3.2. Earth Mover's Distance (EMD) 24
3. RELATED WORK 25
3.1. Neural Architecture Search 25
3.2. Neural Network Pruning 27
3.3. Federated Learning 28
3.4. Federated Neural Architecture Search 30
4. SYSTEM DESIGN 33
4.1. Network Emulated Federated Neural Architecture Search 33
4.2. Neural Network Pruning Algorithm 33
4.3. Federated Learning Framework 35

4.3.1. Flower FL Framework 36
4.3.2. Modification in Flower FL Framework 37
4.3.3. Federated Metric Aggregation 38
4.4. Network Emulator 38
4.4.1. CORE Network Emulator 38
4.4.2. Configurations in CORE 40
4.5. Proposed Client Grouping Algorithm: NetDOG 41
4.6. Overview of NAFNAS 45
5. EXPERIMENTS AND RESULTS 47
5.1. Experiment Settings 47
5.1.1. Dataset 47
5.1.2. Model 47
5.1.3. Resource Type 47
5.1.4. Hardware Settings 48
5.1.5. Network Configuration 48
5.2. Non-IID Dataset Generation of Clients 48
5.3. Experiments of Federated Learning Framework 50
5.4. Experiments of Federated Neural Network Pruning 52
5.5. Experiments of Network-Aware Federated Learning 57
5.6. Network and Distribution Optimized Client Grouping 63
6. CONCLUSION 69
REFERENCES 71

LIST OF FIGURES

Figure 2.1. An overview of AutoML pipeline. 6

Figure 2.2. Overview of neural architecture search. 7

Figure 2.3. Overview of neural network pruning steps. 9

Figure 2.4. Algorithm flow of NetAdapt. 12

Figure 2.5. A model of Fog Computing. 14

Figure 2.6. Steps of Federated Learning. 17

Figure 2.7. FedAvg algorithm. 19

Figure 2.8. FedProx algorithm. 20

Figure 2.9. Sample of IID and Non-IID data distribution. 21

Figure 2.10. Affect of α on class distributions. 23

Figure 4.1. Pruning of AlexNet architecture. 34

Figure 4.2. Original and federated pruning pipelines. 35

Figure 4.3. Flower core framework architecture with edge clients. 36

Figure 4.4. User interface of CORE. 39

Figure 4.5. Architecture of CORE. 40

Figure 4.6. NetDOG algorithm. 42

Figure 4.7. Groups generated by NetDOG in different COEFF settings 43

Figure 4.8. Overall NetDOG System Architecture. 45

Figure 4.9. Neural Network Pruning in The System. 45

Figure 4.10. FL Framework in The System. 46

Figure 5.1. Class distributions and number of samples in α configurations. . . 49

Figure 5.2. Number of samples of each class in each client. 50

Figure 5.3. Aggregated accuracies of Alexnet in federated learning settings. . . 51

Figure 5.4. Reduction in MACs. 53

Figure 5.5. Reduction in model size (MBytes). 53

Figure 5.6. Reduction in model parameters. 54

Figure 5.7. Change in accuracy. 54

Figure 5.8. Federated fine-tuning of models generated in iterations. 55

Figure 5.9. MAC and accuracies of selected models in iterations. 56

Figure 5.10. Groups generated by basic grouping. 57

Figure 5.11. Communication and fine-tuning durations in 1st iteration. 58

Figure 5.12. Overall round durations in the basic grouping. 59

Figure 5.13. Groups generated by Network-aware grouping. 60

Figure 5.14. Federated fine-tuning accuracies with network-aware grouping. . . 61

Figure 5.15. Overall round durations in the network-optimized grouping. 62

Figure 5.16. Groups generated by NetDOG. 63

Figure 5.17. Overall round durations in the NetDOG grouping. 63

Figure 5.18. Comparison of one round durations across grouping methods . . . 64

Figure 5.19. Change in EMD of groups 65

Figure 5.20. Federated fine-tuning accuracies with NetDOG. 66

LIST OF TABLES

Table 4.1. Comparison of EMD values of Different COEFF values. 44

Table 5.1. Bandwidth tiers of clients. 48

Table 5.2. EMD values of basic groups. 58

Table 5.3. EMD values of network-aware groups. 60

Table 5.4. Comparison of EMD values for grouping approaches. 67

Table 5.5. Comparison of validation accuracy across pruned models. 67

LIST OF SYMBOLS

Acc	Accuracy
bw_i	Bandwidth tier of c_i
c_i	Client i
C	Set of all clients
C_j	Subset of C where the clients with group number j
$Dist_{c_i}$	Class dist. of c_i
$Dist_{G_j}$	Class dist. of the subset of G
$Dist_{g_i}$	Class dist. of the group of c_i
$Dist_G$	Class dist. of overall dataset
$\mathcal{D}_{train}$	Train Data
$\mathcal{D}_{val}$	Validation Data
EMD	Earth Mover's Distance
g_i	Group no. of c_i
g_{temp}	Temporary variable for client groups
G	Set of all groups
G_j	Subset of G where the clients with group number j
G_{temp}	Temporary variable for all groups
IRR	Initial resource reduction ratio
$\mathcal{L}$	Objective Function
$p_i^{(k)}$	Proportion of the samples of class i at client k
$P_{y=i}$	Proportion of class i
$\mathcal{R}_{current}$	Current resource
$\mathcal{R}_{target}$	Target resource
RRD	Resource Reduction Decay
w	Window length
ζ	Resource Constraint

LIST OF ACRONYMS/ABBREVIATIONS

API	Application Programming Interface
AutoML	Automated Machine Learning
CNN	Convolutional Neural Network
CONV	Convolutional
CORE	Common Open Research Emulator
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
DP	Differential Privacy
EMD	Earth Mover's Distance
FC	Fully-Connected
FedAvg	Federated Averaging
FL	Federated Learning
GAN	Generative Adversarial Network
GDPR	General Data Protection Regulations
GPU	Graphics Processing Unit
HPC	High-Performance Computing
IID	Independent and Identically Distributed
IoT	Internet of Things
KD	Knowledge Distillation
MAC	Multiply-Accumulate operations
ML	Machine Learning
NAS	Neural Architecture Search
PDF	Probability Density Function
RL	Reinforcement Learning
RNN	Recurrent Neural Network
WAN	Wide Area Network

1. INTRODUCTION

Deep learning (DL) has enabled remarkable progress to be inevitable in our daily lives by revolutionizing our daily lives and the services people are using. DL is enhancing healthcare through accurate medical image analysis, financial security with fraud detection algorithms, and powering technological advancements like natural language processing for virtual assistants and language translation services with the help of smart devices. Moreover, the Internet of Things (IoT), which is the networking of physical devices, that are connected and smart, embedded with software, sensors, and network connectivity, allowing them to collect and exchange data [1], devices play a crucial role in leveraging DL powered services by acting as data sources and endpoints for intelligent applications. Cooperation of DL and IoT devices, enhances the capabilities of IoT systems, enabling them to make informed decisions, predict patterns, and respond dynamically to changing conditions.

The success of DL relies on automating engineering efforts by developing learning algorithms and architectures for Deep Neural Networks (DNN) [2]. The design of DNNs includes various aspects such as network architecture, weights, and hyper-parameters. Only when all those aspects are optimal the DNN performs admissible results for the given task and data. It is widely studied that deep architecture of DNNs is able to extract meaningful information from the data without the help of the special feature engineering effort [2]. However, developing those systems manually requires extensive expertise in DL, experience in similar tasks, and time for investigating various architectures and hyper-parameter settings [3]. Because of this fact, Automated DL (AutoDL) methods emerged to automate deep learning pipelines, especially in finding optimum neural architectures. Neural Architecture Search (NAS) automates architecture search of DNNs by using various methods such as reinforcement learning [4], evolutionary algorithms, and supernetworks, the term NAS coined by Zoph et al. in [4]. NAS techniques have proven superior to manually created structures in a variety of tasks [5]. However, NAS algorithms usually run multiple iterations and require lots of compu-

tational power and time if they are using discrete methods that include individually training and evaluating models in the search space.

The IoT systems have multiple edge devices on the field with data gathering and processing functions and each device stores their data data in local database systems that are named as data-silos. Those devices and systems are being used in various domains from healthcare [6], agriculture [7], intelligent transportation [8] to smart cities [9]. These advancements came with drawbacks that cause data privacy concerns about sensitive information; in 2015, researchers revealed that over 68,000 pieces of medical information were leaked online, where a single healthcare organization has nearly 12,000 of them [10]. To preserve DL capabilities while overcoming data privacy issues, Federated Learning (FL) has emerged. Federated learning is a Machine Learning (ML) approach that complies with the General Data Protection Regulations (GDPR) [11] that the European Union enforced on May 25, 2018, by the contribution of several clients to develop a shared, robust machine learning model without exchanging data. While traditional DL approaches usually use centralized methods to store data in data centers and train DNNs in there, FL has revealed a new decentralized method to address the benefits of data silos and distributed training [12]. When compared to centralized training on persisting data, federated learning offers clear privacy advantages [13]. In a regular Federated Learning scenario, there is a server S with ML model called global model W' and there are N clients $\{C_1, ...C_N\}$ have their local datasets $\{D_1, ...D_N\}$, where any of clients have access to database of others. The server sends global model W' to all of the clients, client C_i trains its own received model W_i with own dataset D_i and sends W_i to server without sharing local dataset D_i, the server aggregates all client models $\{W_1, ...W_N\}$ to update global model W' where it is going to be distributed to all clients.

IoT devices generally have a limited computational budget (latency, memory, FLOPs, etc.) and especially changing network conditions (bandwidth, delay, jitter, location, etc.) while DL-powered services on IoT devices usually require real-time decision-making [14]. These advancements increasingly raised the importance of com-

putational efficiency and data privacy while maintaining the model performance for use cases, which are shaping the ongoing evolution of deep learning applications in our interconnected and data-driven world. Since IoT devices or edge devices have limited resources, DNNs should be feasible for those devices in terms of computational cost and inference latency. Even though traditional DNN architecture design has the objective of increasing the test accuracy, when the computational limitations of edge devices have been considered, another design objective should be the computational cost and size of the DNNs. It is a very challenging issue for developers to achieve those objectives together, therefore network pruning and efficient neural architecture search methods have been developed to find the most feasible DNNs for devices with various computing capabilities. While regular neural architecture search algorithms aim to maximize accuracy, efficient neural architecture search algorithms focus on finding the most efficient DNN architectures and preserving accuracy. On the other hand, network pruning algorithms directly shrink the DNN layers, mainly reducing the layer sizes of DNN to minimize the computational cost and inference latency, to find lightweight architectures while preserving the accuracy at acceptable levels.

However, while aiming to find DNN architecture with the best accuracy or the most lightweight model, researchers may not have access to all data in terms of the private data of users of devices that do not prefer to share. That situation raises the complexity of the architecture search problem. Since federated learning is proposed as a solution for data privacy challenges, executing NAS algorithms in distributed settings is a beneficial solution when architecture search with local data is desired. There are several approaches to architecture search problems and since their search space and search algorithm are specialized in centralized scenarios, federated learning can not be used in NAS straightforwardly. Either NAS and network pruning algorithms have been configured for federated scenarios or algorithms that have been specialized for federated settings have been proposed.

Even though federated neural architecture search and federated network pruning algorithms and systems help find successful lightweight models, such algorithms with

tremendous computation costs cause communication and computation costs on clients, and network conditions of edge and server devices have effects on federated neural architecture search. Therefore effects and limitations in terms of network perspective should be investigated and, as far as we know, it is not widely studied.

On the other hand, data distribution of edge devices is another factor in federated learning systems. In a classification task as an example, it can not be expected that all the edge devices, that in the actual world, have similar data distributions in terms of the sample sizes of each class. More specifically, in an example image classification task where there are 10 classes in the dataset, some edge devices may not have collected any samples in class 1 and class 3, or the edge device may have more samples for specific classes. Since edge devices train the shared model with their local data in federated learning, the locally trained models are affected by data distributions in terms of accuracy, over-fitting, and model weights. Different sample sizes in those data distributions also affect the training duration on edge devices with limited computational resources. Therefore, data distributions would be another important factor in federated learning and also federated neural architecture search and network pruning to find the best-performing architectures.

In this thesis, we focus on the effects of network conditions federated neural network pruning systems and possible solutions for computation cost, communication cost, and convergence time. Moreover, we have set up a fully open-source federated neural network pruning system with an emulated network. Thus, main contributions of our study can be summarized as:

- Our first significant contribution is the development of an open-source federated neural network pruning framework with network emulation support. The framework, named NAFNAS, is specifically built to address the challenges associated with model pruning in a distributed network of clients within federated learning settings and various network characteristics. By efficiently reducing the size of neural networks, our framework tackles the complexities of transmitting and

updating large DNN models with different architectures on resource-constrained devices. Through rigorous testing and experimentation, we have demonstrated the feasibility of our federated neural network pruning approach while maintaining optimal model performance.

- The second key contribution is a novel client grouping algorithm, called NetDOG, tailored for federated learning in an environment of models with different DNN architectures. This algorithm has been developed to account for both the non-identically distributed (non-IID) characteristic of the local data of the client and the limitations caused by network bandwidth constraints. By optimizing the placement of clients into groups based on their class distributions and network bandwidth tiers, our algorithm improves the efficiency in communication round duration and balance of federated learning updates.
- Our third main contribution is an in-depth study of how network bandwidth affects federated learning, taking particular interest in the length of communication rounds. After conducting comprehensive testing, we revealed the connection between changes in network bandwidth and the effectiveness of federated learning processes. To mitigate the consequences of low network bandwidth, this research offers useful insights that enable the design of more resilient federated learning systems.

2. BACKGROUND

2.1. Neural Architecture Search and Neural Network Pruning

2.1.1. Neural Architecture Search

Traditional approaches to developing strong models from a dataset are akin to creating art, requiring intuition, experience, and a slew of time-consuming manual chores to fine-tune algorithmic parameters. Constructing a well-performing machine learning pipeline (including pre-processing, machine learning algorithm selection, and hyperparameter optimization) is a boring and error-prone task for scientists or developers [15]. The idea of automating ML jobs arose from a mix of market pressure for additional ML developers and the time-consuming difficulty of producing 'optimal' ML solutions [16]. Therefore, Automated Machine Learning (AutoML) emerged and it is a concept that reduces repetitive tasks and automatically performs model development processes.

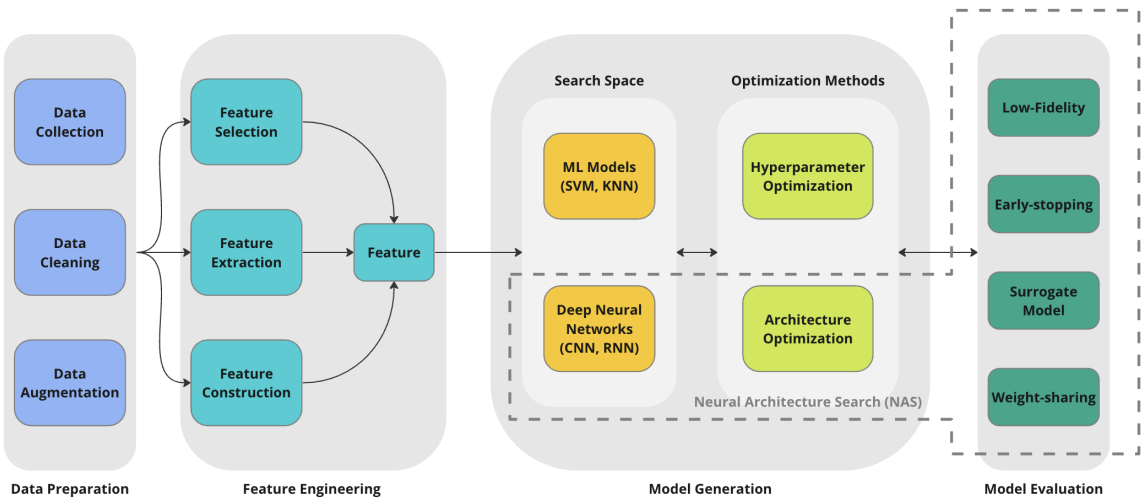


Figure 2.1. An overview of AutoML pipeline.

The AutoML process involves several steps including Data Preparation, Feature Engineering, Model Generation (with hyperparameter and architecture optimization), and Model Evaluation. Data preprocessing and feature engineering are the first steps,

which require human interaction. Models are built and trained using extracted features and datasets, selecting hyperparameters and architectures, illustrated in Figure 2.1. Hyperparameter optimization methods include grid search, random search, and Bayesian search [16]. Performance assessment is crucial, with training on the training set and estimating on the validation set [17].

Neural Architecture Search (NAS) is a subfield of AutoML that provides an alternative for manually designing Deep Learning architectures by automating the process of finding the most suitable DL architectures for a given task. There are multiple architectural components to select and optimize while designing a DL architecture such as layers, layer sizes, components, activation functions, etc. The general idea behind NAS is to build a *search space* that includes various architectures by using the components as a traditional DL approach and selecting the best architecture from that candidate architectures in search space. The selection is based on the search strategy whose objective is maximizing the evaluation objective. The objective in search strategy is generally model performance in terms of accuracy in some studies there would be multiple objectives such as the efficiency [18] or inference latency [19].

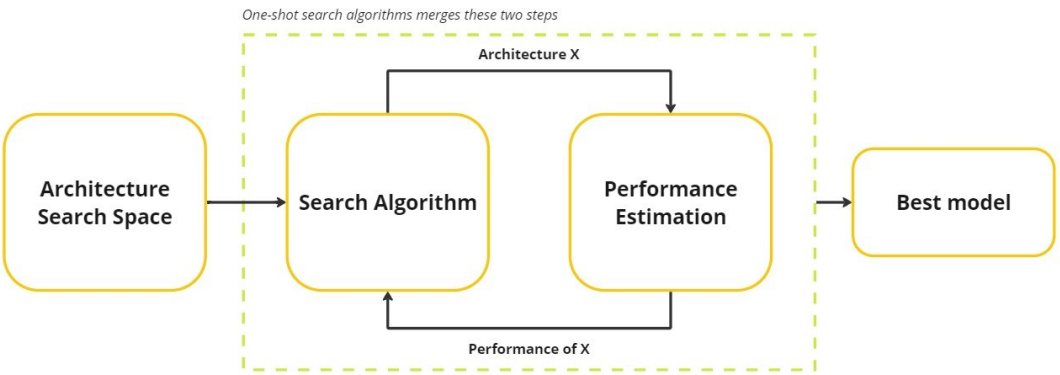


Figure 2.2. Overview of neural architecture search.

Fundamental NAS papers categorize architecture steps in 3 steps: *search space*, *search strategy*, *performance estimation* as illustrated in Figure 2.2. However latest research in one-shot search algorithms does not fit those steps directly, these researchers developed search strategy performance evaluation as one single step [20]. These con-

cepts have been discussed below.

Search space is a set of proposed architectures where the search strategy in the NAS algorithm is going to find the best model. Since the search space concept is a general and wide term, there could be thousands of architecture could be proposed in it. Even though domain knowledge for the addressed task can be beneficial for reducing the size of the search space, too much human effect can cause a bias in NAS which reduces the possibility of finding generalized architectures for a given task [21].

Search strategy is the algorithm for optimizing the finding of the best-performing architecture in the given search space. There are multiple approaches in the search strategy mainly categorized as black-box methods such as Bayesian optimization, reinforcement learning, or evolutionary search; and one-shot search algorithms such as super-net and hypernet-based approaches.

To improve the optimize the search algorithm, the performance of the proposed models needs to be estimated. *Performance estimation* is the technique for estimating this performance; even though the most straightforward option is evaluating the validation performance of these models individually on unseen data, that option can be costly. Researchers are working on new ways to save time and computing resources. When an architecture is proposed, the focus is on how accurately it predicts outcomes on a validation set. But now, researchers are also looking at other objectives, such as the size of the model and how quickly it responds. This is important because some devices have memory limits or need quick responses, so various aspects of a model's performance of models should be considered.

The NAS problem is formulated as [22]

$$\begin{aligned} f^* &= \arg\min_{f \in \mathcal{F}} Cost(f(\theta^*), \mathcal{D}_{val}), \\ \theta^* &= \arg\min_{\theta} \mathcal{L}(f(\theta), \mathcal{D}_{train}), \end{aligned} \tag{2.1}$$

where f^* is the found architecture with its parameters θ^* and $Cost(.,.)$ for a given

search space F, the input data D divided in $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$, and the cost function $Cost(.)$, we aim at finding an optimal neural network $f \in \mathcal{F}$, which could achieve the lowest cost on dataset $\mathcal{D}$.

2.1.2. Neural Network Pruning

The breakthrough performance of DNNs in perceptual real-world tasks highly relies on a very large number of parameters and computation power [23]. Therefore there is a positive correlation between improving accuracy and required computation power. However state-of-art DNN models require high computational resources from the devices they are executed while those requirements far exceed the computation resources of many devices on edge and embedded devices. Therefore, methods for reducing the computational requirements of a DNN model while maintaining accuracy are required.

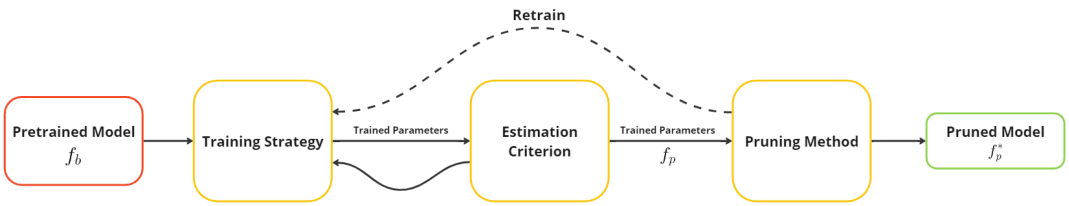


Figure 2.3. Overview of neural network pruning steps.

At that point, neural network pruning and model compression algorithms have emerged to build lightweight models for devices with low resource limits. There are various approaches at this end such as tensor decomposition [24], parameter quantization [25], and neural network pruning generally categorized as structured pruning and non-structured pruning. The general procedure of neural network pruning is illustrated in Figure 2.3.

Network pruning is a strategy for reducing deep neural network model complexity while retaining accuracy. It implies removing the network of redundant connections and weights to provide a lighter model that can be used on embedded and edge sys-

tems, which have limited resources. Feature shift minimization, distribution-lossless pruning, and matrix sparsification are a few examples of criteria that may be used to guide pruning techniques. By identifying and eliminating unnecessary or insignificant network components, these techniques seek to create a more effective and interpretable model. It has been demonstrated that neural network pruning may effectively reduce overparameterized models without appreciably sacrificing accuracy.

Neural network pruning is beneficial for several reasons. Smaller models take up less space on storage devices, making them ideal for deployment on resource-constrained platforms such as mobile phones or embedded systems. With fewer connections and neurons to compute, the model runs faster during prediction. This is crucial for real-time applications where quick responses are essential. Moreover, lightweight models require less computational power, leading to lower energy consumption during training and inference. Pruning can be tailored to the specific hardware or software platform where the model is going to be deployed. For example, a model deployed on an edge device might need to be aggressively pruned to fit within the memory constraints of the device.

Pruning techniques used for Convolutional Neural Networks (CNNs) include kernel pruning and non-structured weight pruning. Setting each weight to zero and measuring the error change is a brute-force pruning technique. Remove it if it grows too much; if not, return the weight [26]. Weight pruning involves zeroing unnecessary connections in the weights of convolutional layers, while kernel pruning trims the kernels of the same input feature map simultaneously for structural consistency. CNN pruning methods can be categorized based on *pruning method*, *training strategy*, and *estimation criterion* [23] similarly in NAS.

Pruning method decides the trained parameter set by which component to prune and how to prune. In order to minimize computation during execution, non-structured pruning is a technique for CNNs in which unnecessary connections or weights are zeroized. However, for the kernel, which is a small matrix component that extracts

features in CNN models, weight zeroing can not be applied because of the structural dependency of the CNN architecture that needs to be preserved. After a kernel is zeroized and then removed from the network structure without considering the individual connections of kernels with other components, the structure of the CNN would collapse.

On the other hand, *structured pruning* removes the entire components from the deep neural networks to reduce the size and computational cost of DNNs. For the context of CNNs, ineffective or unnecessary filters are eliminated from the model by structured pruning without affecting the architectural stability of the CNN model. Filter pruning is a very common approach where filters that contribute less to the overall network performance are then pruned or removed, moreover, it is crucial to calculate the importance of filters that are desired to be pruned. As an example, in [27], the importance of each filter is based on statistics information computed from its next layer, rather than its layer.

As mentioned above, one of the advantages of neural network pruning is the benefit for adapting the DNNs to resource-constrained devices. The resource constraint filter pruning can be formulated as

$$\begin{aligned} f^* &= \arg\min_{z,\theta} \left| \mathcal{L}(f(\theta^*)) - \mathcal{L}(f(\theta \odot z)) \right|, \\ C(z) &\leq \zeta, \end{aligned} \tag{2.2}$$

where $\mathcal{L}$ is the objective function, f is the DNN model, $\theta = \{\theta_1, \theta_2, ..., \theta_K\}$ is the target model weights and $\theta^* = \{\theta_1^*, \theta_2^*, ..., \theta_K^*\}$ is the pre-trained model weights where K is the dimension of the weight vectors which each dimension is a filter. When looking at the constraints, $C(\cdot)$ is the target resource usage and ζ is the resource constraint. $|\mathcal{L}(f(\theta^*)) - \mathcal{L}(f(\theta \odot z))|$ is the loss difference caused by z.

There are several approaches to the resource constraint neural network pruning problem, one of them is called NetAdapt [28]. NetAdapt is an algorithm that automatically simplifies a pre-trained deep neural network to meet the resource budget of a

platform while maximizing accuracy. It incorporates direct metrics, such as latency and energy consumption, and therefore does not require platform-specific information; also in the paper of NetAdapt, the number of multiply-accumulate operations (MACs) is set as the constraint in many experiments. After the resource is selected, the NetAdapt algorithm splits the resource constraint neural network pruning problem into smaller easier steps and solves them iteratively. It enables a resource reduction schedule

$$\begin{aligned} f_i^* &= \arg\min_{f_i} Cost(f_i), \\ \text{s.t. } C_j(f_i) &\leq C_j(f_{i-1}) - \Delta R_{i,j}, \end{aligned} \tag{2.3}$$

where the objective is finding f_i^* at iteration i^{th}. f_i is the network generated in i^{th} iteration, and f_0 is the pretrained DNN model. The current resource budget denoted as $C_j(f_{i-1}) - \Delta R_{i,j}$ becomes tighter when the iterations increase. $R_{i,j}$ defines how much the constraint becomes tighter for j^{th} resource at i^{th} iteration.

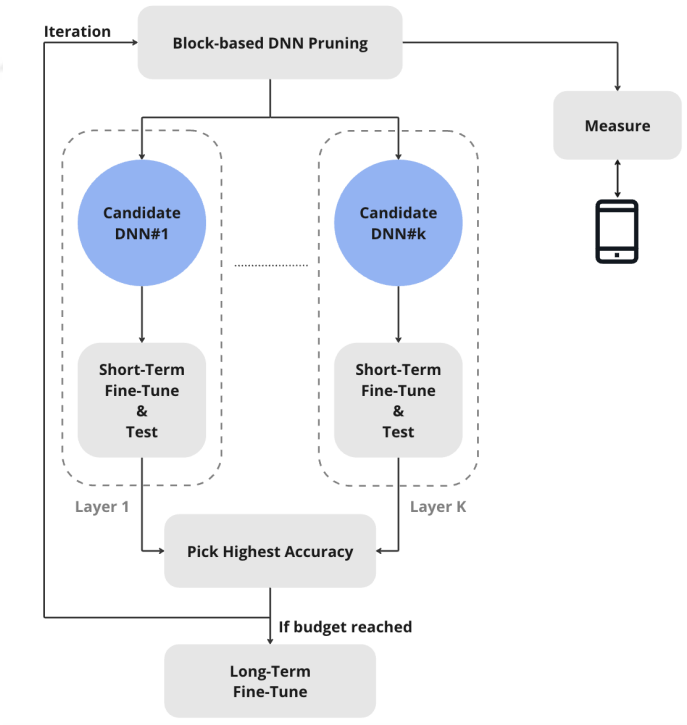


Figure 2.4. Algorithm flow of NetAdapt.

The NetAdapt algorithm selects convolutional layers (CONV) and fully connected layers (FC) to fit in the resource constraint. The algorithm consists of 3 steps in pruning

iterations; (1) choosing the number of filters, (2) choosing which filters to preserve, (3) short-term fine tuning; illustrated in Figure 2.4.

The first phase entails determining the optimal number of filters to preserve in a specific layer through empirical measurements. Each layer of the network is investigated individually and there is a gradual reduction in the number of filters within the target layer. Reduction is decided by by an evaluation of the resource utilization of simplified networks and the candidate child architecture with a maximum size of filters is selected. In the filter selection phase, it is decided which filters to retain, building on the architecture determined in the previous step. To keep things straightforward, a magnitude-based method is applied. Specifically, N filters with the highest 2-norm magnitude are kept, where N corresponds to the number of filters established in the preceding step. The fine-tuning process in NetAdapt consists of short-term and long-term phases, during each iteration, the pruned networks undergo a relatively brief fine-tuning process, with the aim of restoring accuracy. In the measurement of resources such as inference latency, the algorithm generates a latency lookup table beforehand by measuring the inference latency of different layers with their sizes. In the pruning process, the algorithm just checks the look-up table whether the resource of the pruned model meets the constraint.

2.2. Federated Learning

2.2.1. Cloud Computing and Data Management

In this section, cloud computing and data management techniques and approaches are discussed such as centralized and distributed architectures from data privacy and network topology perspective.

Cloud computing is a technology that enables users or resource-limited devices to benefit from and utilize computing resources over the network to reduce the cost and increasing efficiency. It is a breakthrough technology that enables secure data storage

and communication between edge devices and data centers. Previously, only cloud computing technology existed, and multiple cloud service providers offered clients a variety of cloud services.

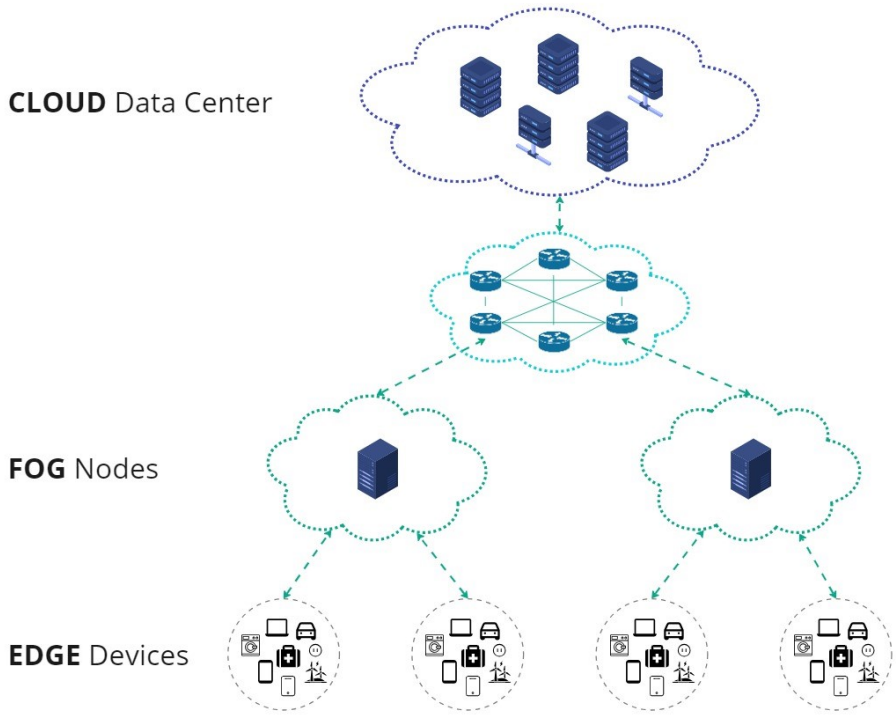


Figure 2.5. A model of Fog Computing.

Edge computing is a distributed computing technique that places the processing of data near the sources of data generation. Edge computing is required when immediate action, low-latency responses, and data processing are crucial on edge devices such as IoT devices since it minimizes the limitation of transmission over long distances to centralized servers.

Fog computing enhances the principles of edge computing by the location of computational resources and services near the edge of the network. Fog computing emerged to address the trade-off between the advantages of low-latency responses of edge computing and the computational capacity of cloud computing. Fog computing optimizes data processing, storage, and analytics across network layers, reducing latency and enhancing bandwidth efficiency and scalability, ideal for applications requiring a mix of

local and centralized processing. A model of fog computing is illustrated in Figure 2.5.

Moreover, *cloudlets* are small-scale cloud data centers situated at the edge of a network, integral to the concept of fog computing. A cloudlet provides a localized computational resource for specific tasks and is located close to the edge devices. When a mobile edge device moves from one region to another, it connects from one cloudlet to another. Therefore, it reduces latency and enhances the efficiency of fog computing architectures.

Even though using cloud computing to handle resource-intensive applications on remote servers for edge devices is a potential solution, the obstacle of long WAN latency remains. When focusing on large tasks like processing high-quality images, it can take some time because the data has to go through the network. The speed at which the data moves, known as bandwidth, determines how long the process takes. So, the delay is not just about processing; it is also limited by how fast data can travel through the network [29].

On the other hand, beyond computational resource advantages in cloud computing concepts, another concept is data management in that kind of distributed system. One of the most straightforward approaches in data management is centralized management which is one of the most easy-to-deploy approach. In a centralized data management system, all the data is stored and processed in a data center or server and the data center is responsible for the management of those tasks. However, the leakage of sensitive data can be a crucial issue in terms of data privacy. Therefore, distributed approaches have become more popular to handle that problem.

Distributed data management emerged to overcome problems such as data privacy and, from computing perspective, effective and fault-tolerant data processing. In such a system, all the data is spread among servers or clients in multiple locations. Since each node is responsible for managing local data and processing, there may be some complexities related to data consistency and coordination between nodes.

2.2.2. Federated Learning

ML models are being used in various tasks of daily life, and being utilized in different types of devices such as computers, data centers, mobile phones, and IoT devices. If the model training process relies on a single center, there is a risk of violating sensitive data privacy because all the data used to train the model needs to be present on that central server, in other words, a client may not be willing to share their own data. In 2016, Google proposed the Federated Learning solution [30] which spreads the data management and model training processes to individual clients or participants in order to maintain data privacy.

Federated learning represents a machine learning methodology wherein the model undergoes training across numerous decentralized devices or servers, each retaining its local data samples without the need for data exchange. In a regular federated learning system, each client, which can be an edge device mobile phone, etc., utilizes their own dataset to train a globally distributed ML model, in our case a DNN model, services by a server. The server coordinates the federated learning process for training and evaluation steps and gives required commands to the clients. A round is defined as the sending global model to participant clients and receiving locally trained model from clients to aggregate and define a new global model, then sending the new global model, the aggregated model, to clients again for the evaluation step and collecting the performance measurements such as accuracy, latency, etc. In detail, the server sends an estimate of DNN model parameters to clients. Clients build the model with received weights and train the model to update the parameters with the local dataset, then send the updated weights to the server. The server waits to collect sent weights from all participant clients and then aggregates with an *aggregation algorithm* all weights to build a new global model. The server sends new global DNN model weights to participant clients for the aggregation process. Clients evaluate the new global model with the local dataset and send evaluation results to the server again. It is crystal clear to observe that in each round there are huge amount of communication between the server and clients due to the size of model weights. This approach prioritizes privacy

and security by conducting the training process on a per-user basis, eliminating the need to share data with the server.

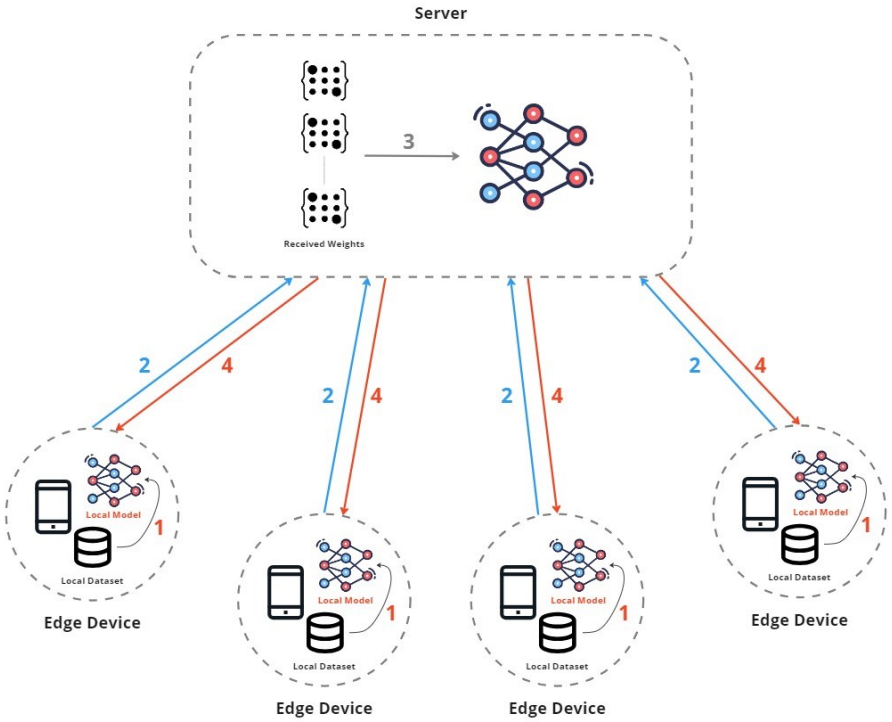


Figure 2.6. Steps of Federated Learning.

The illustration in Figure 2.6 represents the federated learning process; step 1 is the local training process that happens in the client, in step 2 the updated parameters are sent to the client, the server aggregates the received weights in step 3 to build new global model, with step 4 the new global model distributed to the clients for deployment or next round. For the training of the global model, these steps are repeated in multiple rounds.

"Heterogeneous clients" in a federated learning setting are users with different network properties such as network bandwidth. Clients who can participate more frequently and produce better results can be given priority by the server, which could introduce bias into the federated learning process. This undesirable state might result in performance losses, showing how crucial it is to create techniques that preserve clients from bias in a federated learning environment with a range of limitations.

The problem of IID (Independent and Identically Distributed) and Non-IID (Non-Independent and Identically Distributed) dataset distributions in federated learning occurs in the nature of data distribution among the participating clients in the actual world. In an ideal IID scenario, each local dataset of a client is representative of the overall data distribution, ensuring that the model is trained uniformly across all clients. However, in a non-IID setting, clients may have irregularly distributed or non-representative datasets, leading to challenges in aggregating model updates.

2.2.3. Weight Aggregation Algorithms

Weight aggregation algorithms are techniques used in federated learning to combine local model updates from several devices, accounting for factors like data distribution, model performance, and participation frequency. These algorithms seek to create a global model that makes use of the knowledge that clients can contribute.

However, aggregating weights of local DNN models of clients are vulnerable to result in a non-optimal model since the local dataset of clients is generally Non-IID. A certain amount of uncertainty is introduced into the learning model via statistical heterogeneity. Because of that, selecting the right aggregation and learning methodology is crucial for the robustness of FL systems.

Dealing with statistical heterogeneity, or non-IID data, presents challenges in model selection and hyper-parameter tuning due to decentralized data at client locations. The lack of human annotations or interactions with learning systems at the edge clients results in *label scarcity* and limits the application domain of supervised machine learning models [31].

Algorithm Federated Averaging (FedAvg)

Input: No. communication rounds: $\mathcal{T}$, no. epoch in local training: $\mathcal{E}$, no. clients: m, no. selected clients: $\mathcal{C}$, initial model parameters: $w^{(0)}$, model parameters of client c at round t: $w^{(t)}$, set of model parameters received from all participant clients: $w_{\mathcal{S}_t}^{(t)}$, no. samples of client c at round t: $n_c^{(t)}$, set of sample sizes received from all participant clients: $n_{\mathcal{S}_t}^{(t)}$, local dataset of a client: $\mathcal{D}_{local}$

Output: Trained model parameters $w^{(T)}$.

function SERVER_SIDE_OPERATION

Initialize parameters $w^{(0)}$

for t=1 **to** T **do**

Select a random client subset $\mathcal{S}_t$ from m clients

for User c **in** $\mathcal{C}$ **do**

$\mathrm{w}_c^{(t)}, n_c^{(t)} = client_side_operation(\mathrm{w}^{(t)})$

Append $w_c^{(T)}$ to $w_{\mathcal{S}_t}^{(t)}$

Append $n_c^{(T)}$ to $n_{\mathcal{S}_t}^{(t)}$

$\mathrm{w}^{(t)} = FedAvg(w_{\mathcal{S}_t}^{(t)}, n_{\mathcal{S}_t}^{(t)})$ ▷ Weight aggregation

return $\mathrm{w}^{(T)}$

function CLIENT_SIDE_OPERATION(w)

$\mathrm{f}_{local} \leftarrow f(w)$ ▷ build model with received parameters w

for Epoch e **in** $\mathcal{E}$ **do**

Train f_{local} with $\mathcal{D}_{local}$

return $get_parameters(f_{local})$

function FEDAVG($w_{\mathcal{S}_t}^{(t)}, n_{\mathcal{S}_t}^{(t)}$)

$w^{t+1} = \frac{1}{\mathcal{C}} \sum_{c \in \mathcal{S}_t} w_c^{(t)}$

return w^{t+1}

Figure 2.7. FedAvg algorithm.

A lack of labeled data in decentralized environments leads to label scarcity in federated learning. Federated learning strategies control uncertainty from unlabeled data to deal with it. In order to reduce divergence during local training epochs, the server chooses dependable client models from earlier rounds to ensure consistency.

Algorithm FedProx

Input: No. selected clients: $\mathcal{C}$, initial model parameters: $w^{(0)}$, model parameters of client c at round t: $w_c^{(t)}$, set of model parameters received from all participant clients: $w_{\mathcal{S}_t}^{(t)}$, no. samples of client c at round t: $n_c^{(t)}$, set of sample sizes received from all participant clients: $n_{\mathcal{S}_t}^{(t)}$, proximal parameter: γ

Output: Trained model parameters $w^{(T)}$.

function SERVER_SIDE_OPERATION

// Same with FedAvg algorithm

function CLIENT_SIDE_OPERATION(w^t)

$f_{local} \leftarrow f(w^t)$ ▷ build model with received parameters w^t

for Epoch e **in** $\mathcal{E}$ **do**

$\gamma = \|w - w^t\|^2$

$\arg\min_w h_c(w; w^t) = f_{local}(w) + \frac{\mu}{2} + \gamma$ ▷ Objective Function

return *get_parameters*(f_{local})

function FEDPROX($w_{\mathcal{S}_t}^{(t)}, n_{\mathcal{S}_t}^{(t)}$)

$w^{t+1} = \frac{1}{\mathcal{C}} \sum_{c \in \mathcal{S}_t} w_c^{(t)}$

return w^{t+1}

Figure 2.8. FedProx algorithm.

There are several weight aggregation, or in other words model fusion, algorithms that may handle data scarcity or communication costs. Generally, the goal of federated learning the minimize the objective on local data as

$$\min_{\theta} f(\theta) = \sum_{k=1}^{m} p_k \mathcal{L}(\theta), \tag{2.4}$$

where $\mathcal{L}_k$ is the local objective of the k-th client and $\sum_k p_k = 1$. Federated Averaging (FedAvg) [13] algorithm initializes clients randomly at first then distributes the weights to train locally on clients. The learning objective is constructed as $p_k = \frac{n_k}{\sum_k n_k}$. That parameter acts as a regularization parameter by randomly selecting a sample of participant clients in each round of communication and training. The FedAvg algorithm is described as pseudocode in Figure 2.7.

FedAvg algorithm selects clients randomly for each communication round in FL. However, it does not consider the data distribution, in other words, label scarcity. FedProx [32] is another weight aggregation algorithm for statistical heterogeneity problems. Statistical heterogeneity stands for the variability of data distributions in clients. FedProx is an improvement over FedAvg by adding a *proximal term* to the local training objective of each client. Without a proximal term, the algorithm has no difference from FedAvg. But with the proximal term, FedProx guarantees convergence in clients with non-IID data heterogeneity. The proximal term is calculated as

$$\gamma = \left\| w - w^t \right\|^2, \tag{2.5}$$

where w is the parameters updated in the client with its local dataset as described in Figure 2.8, w^t is the parameters received from the server (global model).

2.3. IID and Non-IID Data Distributions

IID (Independent and Identically Distributed) refers to a characteristic of a dataset where each data point is independent of others, and all data points are created from the same underlying probability distribution. This assumption simplifies statistical analyses and is commonly assumed in traditional machine learning settings. On the other hand, Non-IID (Non-Identically Distributed) characterizes a dataset where data points may have different probability distributions, variability, or heterogeneity in the statistical properties.

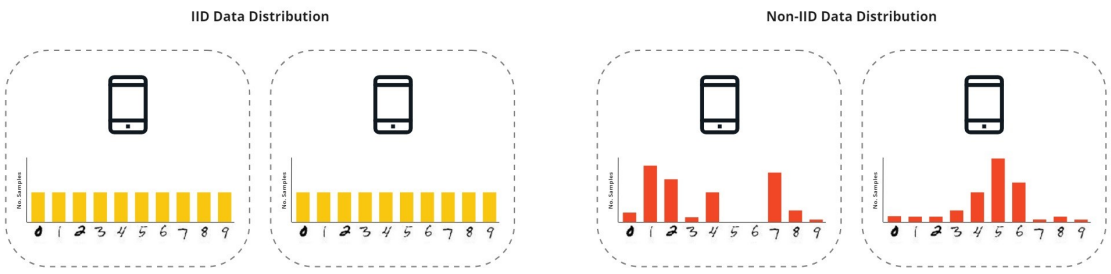


Figure 2.9. Sample of IID and Non-IID data distribution.

Non-IID scenarios which are illustrated in Figure 2.9, are closer, compared with IID data, to real world scenarios, such as federated learning or datasets collected from diverse sources. The non-identically distributed distribution of the data causes the model weights trained on one subset of the data to differ significantly from those learned on another subset, which is called *weight divergence*. In the context of federated learning, local datasets of clients with unique statistical properties would cause the weights to diverge differently therefore that situation would prevent the weight aggregation algorithm from converging or generalizing. Zhao et al. [33] showed the weight divergence of a federated system on clients with a Non-IID dataset.

In federated learning research, there are a couple of ways to distribute experimental datasets to clients in a non-IID manner. In a classification task, one is assigning each client one or multiple image classes and utilizing only samples from selected classes to clients, Non-IID(1) represents each client having data from 1 selected class and Non-IID(2) is for 2 selected classes. Another approach is using statistical approaches that guarantee the statistical properties of distributed data as Non-IID such as *Dirichlet Distribution*.

One assumption is that the used overall dataset, which is split and distributed to the clients, has a uniform class distribution. However, in a real-world setting, there may be other types of class distributions or some unbalanced data problems. In that case, the global model can be biased towards the majority classes. To overcome that situation, class balancing methods such as resampling or synthetic data generation methods can be applied with a strategy on clients that is guided by the server.

2.3.1. Dirichlet Distribution

The Dirichlet distribution is a multivariate continuous probability distribution. The Dirichlet distribution is parameterized by a K-dimensional vector $\alpha = \alpha_1, ..., \alpha_K$,

where $\alpha_i > 0$ for all i. Its probability density function (PDF) is

$$f(X; a) = \frac{1}{B(a)} \prod_{i=1}^{K} x_i^{a_i - 1}, \tag{2.6}$$

where $B(\alpha)$ is a normalizing constant and $\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$ is the gamma function.

Samples from the Dirichlet distribution are K-dimensional vectors. The support is the open (K-1)-simplex, defined as

$$x \in RK : x_i \in (0, 1) \text{ for all } i = 1, 2, ..., K; \sum_{i=1}^{K} x_i = 1. \tag{2.7}$$

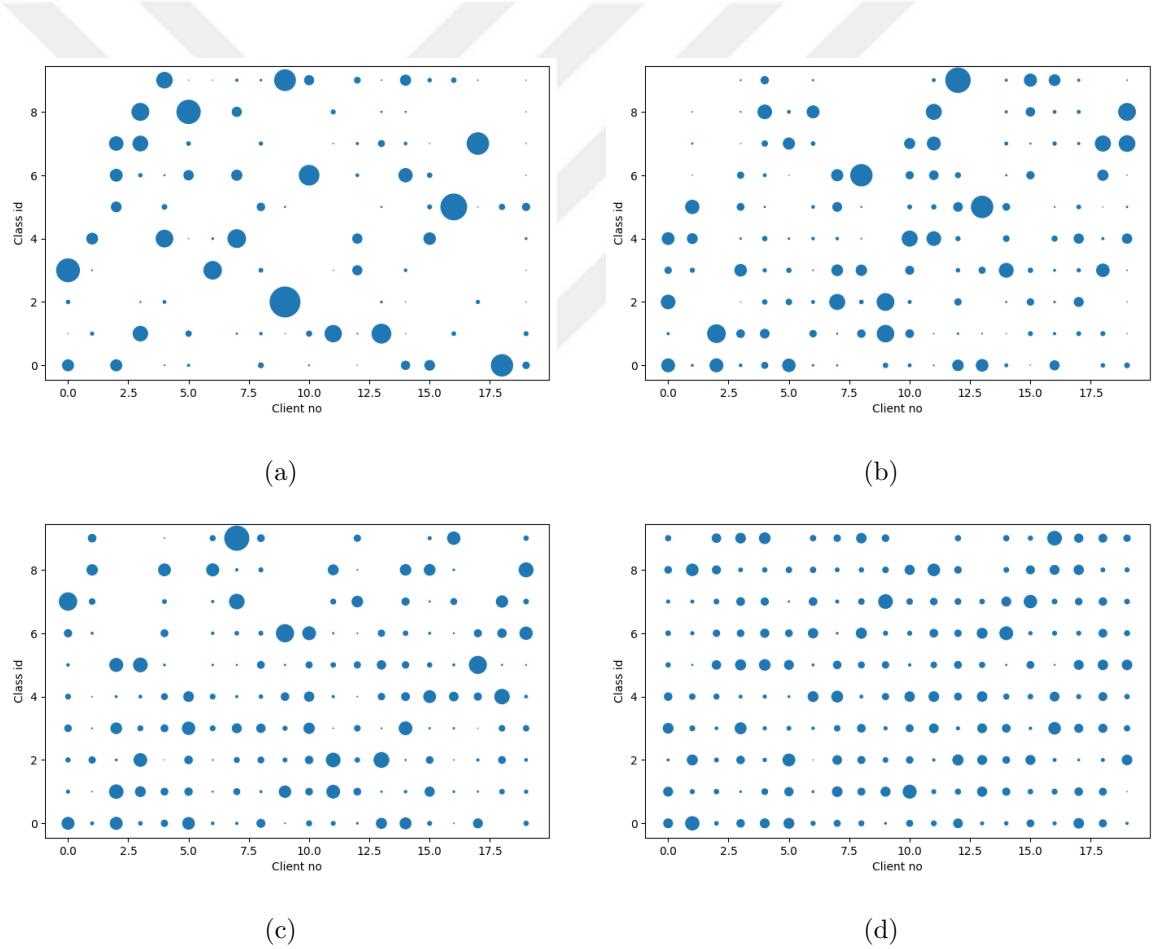


Figure 2.10. Affect of different α values on class distributions where α equals to (a) 0.1, (b) 0.3, (c) 0.5 and (d) 1.0.

The Asymmetrical Dirichlet distribution is employed to model a situation where datasets are distributed unevenly among agents. This is achieved by assigning samples within each class unevenly across agents, using proportions sampled from a symmetrical

Dirichlet distribution. If we have a dataset with C classes and the goal is to distribute it among K agents, a symmetrical K-dimensional Dirichlet distribution $Dir(\alpha)$ is defined, with α as the parameter set. Vectors are then sampled from this distribution for each class, determining the allocation of data from each class to each agent. The α parameter is tunable to regulate the level of uneven distribution. Higher α values result in more balanced distributions, while lower values lead to more uneven splits [34].

The effect of α value on sampling with Dirichlet distribution is illustrated on the CIFAR-10 dataset with 20 clients in Figure 2.10(a), Figure 2.10(b), Figure 2.10(c) and Figure 2.10(d).

2.3.2. Earth Mover's Distance (EMD)

The Earth Mover's Distance (EMD), also known as Wasserstein distance, is a metric that measures the similarity of two distributions defined in the same space. The fundamental approach in EMD is determining the minimum work required to transform one distribution into another. In [33], the EMD is stated as the root cause of the weight divergence that results in low accuracy performance in the validation dataset and formulated as

$$EMD_k = \sum_{i=1}^{C} ||p_{y=i}^{(k)} - P_{y=i}||, \tag{2.8}$$

where C denotes the number of class types, $p_{y=i}^{(k)}$ refers to the proportion of the number of the i^{th} type data of client k and $p_{y=i}$ refers to the global proportion. It is already known that higher EMD values across the clients lead to a higher weight divergence between locally trained models and lower validation accuracy on the aggregated global model [33]. Authors have proposed a data-sharing strategy from server to client to balance the distributions but that solution requires a common dataset on the server.

3. RELATED WORK

3.1. Neural Architecture Search

Neural Architecture Search is the process of building the best-performing neural network architecture for a given task in an automated way. NAS has already shown success in various tasks [35–37]. One of the prior works [4] by Google Brain utilized Reinforcement Learning (RL) in the neural architecture search process. The proposed method uses Recurrent Neural Networks (RNN) for architecture construction and RL to maximize the accuracy of a validation dataset. Notably, RNN proposes a new network architecture by acting as a sequence generation task. If the generated model performs well controller gets a reward otherwise, it gets penalized, where the controller in RL learns expected validation errors and proposes better architecture in the following iterations. The proposed method has been tested on CIFAR-10 and Penn Treebank datasets and used 800 GPUs for two weeks to achieve SOTA results. By realizing there are certain motifs in architecture engineering Zoph et al. [38] defined two types of convolutional cells for architecture building; one returns a feature map in the same dimension other reduces the dimension. That approach costs 500 GPUs and 4 days. ENAS (Efficient-NAS) [39] used a similar approach but used weight sharing to speed up the reward estimation. The search space has been effectively reduced by sharing weights across different architectures, which also allows faster exploration and evaluation of various architectures. ENAS method achieved similar results with only 1 GPU and less than 16 hours.

Early, it was a common approach to use evolutionary algorithms for optimizing the neural network architecture [40, 41]. A population of architectures is updated iteratively by evolutionary NAS methods. Each phase involves selecting one or more parent architectures from the population by its validation accuracy, combining them, and undergoing mutations to produce new child architectures [21]. There are various approaches to sampling the initial population and parent selection from the population.

Real et al. [42] randomly sampled the architectures from the search space and also used regularized evolution that eliminates the architecture in each phase which has been in the population for a long time. On the other hand, Elsken et al. [43] selected the best-performing architectures as parents from the population. LEMONADE [44] is a multi-objective evolutionary NAS algorithm that aims to minimize multiple objectives by considering the trade-off between predicting accuracy and consumption of resources.

RL and evolution-based NAS algorithms iteratively find architectures from search space and train them to obtain their performance. However, these approaches require immense computational costs because of the process of training thousands of architectures individually. Therefore, one-shot NAS techniques have emerged and become popular in the literature. With one-shot techniques each architecture in the search space is not trained from zero, just the opposite all architectures are trained by single training of a *supernetwork*.

A supernetwork is a large network that can build all possible neural network architectures in the search space by containing every possible operation [21, 45]. The supernetwork idea is introduced by Saxena et al. [46]. The method proposed by Bender et al. [45], first trains the supernetwork and extracts the best sub-networks from the trained supernetwork. Once-for-all [47] is proposed to find an efficient neural network architecture to fit in memory and inference latency requirements. Once-for-all uses the progressive shrinking approach to generate neural network architectures with different sizes.

The most popular one in supernetwork-based NAS methods is DARTS (Differentiable Architecture Search) [48] proposed by Liu et al. The DARTS method uses continuous relaxation to represent the search space which makes it possible to use the gradient descent algorithm to extract best performing architecture more rapidly.

3.2. Neural Network Pruning

Convolutional Neural Network (CNN) is a popular branch of DNNs, especially in image processing tasks. Usually, CNNs are over-parametrized [18]. Advancements in the last decade in DL and DNNs and their use cases in edge devices have brought many kinds of research in neural network pruning along. There are various categories in these researches [49]. Magnitude-based pruning methods [50] measure the degree of importance of weights and neurons by their magnitudes. Sensitivity analysis-based pruning [51] investigates the effect of weights on the validation loss by removing or perturbing them. Pruning with knowledge distillation [52] is another approach of neural network pruning where the main model, that is to be pruned, is teaching the more efficient pruned model in the training phase. Quantization methods [53] focus on quantization, binary, and low-precision representation of the DNN model for compression. Also, NAS-based methods [54] have been used to find more compact and efficient DNN models.

A comprehensive pruning method was proposed by Lin et al. [51] to remove unnecessary channels from a deep neural network. Their approach consists of two steps: first, the significance of each channel was measured, which resulted in the removal of unnecessary channels from all layers. Second, the method dynamically modifies filter accuracies by comparing sparse and pruned networks to fix any improperly pruned channels. To accelerate the inference process, Li et al. [55] proposed a refined pruning mechanism where channel-wise significance indexes and layer-wise sparsity of convolutional layers have been focused. Another structured pruning algorithm proposed by Chung et al. [56] where the proposed method prunes specific convolution channels in the first layer of CNN because of the fact that pruning of the first layer lets compressing the rest of the convolution layers successfully.

The foundation of unstructured pruning was a heuristic method for eliminating unnecessary parameters [57]. Han et al. [58] introduced the train-prune-retrain method to detect important connections to reduce the storage and computational cost of DNN.

Yang et al. [59] utilized the energy cost of each layer in order to select and order layers to prune. NetAdapt [28] is another approach in neural network pruning that optimizes the pruning process by considering another objective such as inference latency and computational cost constraints.

Quantization of DNN models is a process of reducing the precision of the numerical values used to represent the weights and parameters of the network. Post-training quantization is a process where quantization of weights of the DNN model, followed by re-optimization to produce a quantized model with scales [60]. Quantization-aware training is a technique to train DNN with the awareness that the model is later to be quantized to lower precision; to integer values, typically 8-bits [61, 62].

Knowledge distillation [63] is a model compression technique where a smaller DNN (student) is trained by a more complex DNN model (teacher). The student network learns various knowledge from the teacher network such as ground truth labels, soft probabilities, or intermediate representations. This method helps transfer the knowledge encoded in the teacher model to the smaller student model, enabling more efficient and compact neural network representations. One method [64] used middle layers to transfer the feature-based knowledge of teacher DNN.

3.3. Federated Learning

Federated Learning (FL) was coined by McMahan et al. [13] as a distributed machine learning (ML) framework in order to preserve data privacy. Because of the advancements in data privacy, it is utilized in various fields such as mobile keyboard prediction [65], financial fraud detection [66], and precision medicine [67]. However, there are challenges in FL from different aspects such as privacy protection, statistical heterogeneity, device heterogeneity, and communication heterogeneity.

ML techniques generally assume that data are independent and identically distributed (IID) [68], however, non-IID data distributions are a problem of data het-

erogeneity. Hsieh et al. [69] investigated the effect of the degree of data skew on the performance of DNN models. FedAvg [13] is proposed to utilize the stochastic gradient descent(SGD) algorithm with a global aggregation setting. Even though, the paper of FedAvg states that the aggregation algorithm is still robust for non-IID data settings, some experiments reveal that FedAvg is insufficient on non-IID data [33]. Itahara et al. [70] shared unlabeled nonprivate data with clients in order to overcome the weight divergence problem that is resulted from non-IID data. FAug [71] is a method to generate missing samples by a GAN generator. Clients send local samples to the server to train GAN and GAN is published to clients for local data sample generation process. Another way to deal with the data heterogeneity problem in FL is by selecting participating clients or training samples across clients and local datasets. FedSS [72] is a dynamic sample selection algorithm that selects training sample size from a local dataset of clients to improve convergence and reduce computational cost. Wang et al. proposed FedACS [73] that is aimed to build an ideal client pool by selecting clients based on their class distribution heterogeneity by statistical analysis. Zeng [74] proposed predicting the DNN model training on heterogeneous devices by a deep learning method in order to investigate the different model features that have a significant effect on training time and then used key features to train a global model. The optimization can be placed into model training or aggregation phase, Ma et al. proposed FedSA [75] a semi-asynchronous FL mechanism that selects a certain number of local models that are going to contribute global model updates by their order of arrival. Reddi et al. [76] used ADAGRAD, ADAM, and YOGI optimization methods in server and client optimizers and implemented those methods in the FedAvg algorithm. The results of that research revealed that an adaptive optimizer can improve the convergence of DNN models significantly. Model regularization optimization is a method for increasing the efficiency and global accuracy in FL [77]. FedProx [32] algorithm adds a proximal term to the local optimization objective to balance the distance between local and global weights with L2-distance.

On the other hand, while the objective of FL is data privacy, it requires communication over the network and there may be some vulnerabilities that need to be handled.

Differential privacy is a concern that is a strategy that prevents attackers from inferring the local dataset from an output [78]. In DP-FedAvg [79], differential privacy is applied to federated learning in client-level data protection by the Gaussian mechanism. [80] applied the DP-FedAvg algorithm to mobile devices. FedMD-NFDP [81] implements a noise-free differential privacy mechanism to a knowledge distillation-based FL method.

Knowledge distillation (KD) [63] is applied in FL in order to alleviate the communication cost and duration issues. The integration of knowledge distillation to federated learning increases the convergence and also it is able to handle model heterogeneity problems since only the logit size is required in KD. Li et al. [82] introduced FedMD which allows clients to train local models on public datasets and upload logits to the server rather than model parameters. That approach requires the generation of a public dataset and therefore may not be feasible for each task. FAS [83] is proposed to increase the convergence and reduce the communication cost. Researchers have analyzed the effect of public data distribution and logit aggregation strategy theoretically and proposed an active sampling method with the DP strategy. FedDistill [84] is proposing a federated distillation method without data privacy problems by forcing clients to share a global model while training a local model. A client has two models one is a global model that is trained to share with the server, other is a local model that serves as a teacher model to train a global one. Wu et al. [85] proposed FedKD that the global model can be trained with three loss functions and matrix factorization is applied to reduce the communication overhead.

3.4. Federated Neural Architecture Search

There is an increasing interest in searching for a neural architecture under federated learning settings because of the demand for privacy-preserving distributed training. Federated NAS methods can be categorized as offline and online Federated NAS [86]. In offline Federated NAS, the architecture search process is conducted in a centralized server and the candidate DNN architecture is distributed to the participant clients for local training and deployment. Zhu et al. [87] proposed a genuine offline

federated NAS framework where an evolutionary optimization algorithm for selecting optimal architecture from the Pareto front. On the other hand, in online federated NAS, the search for the optimum of the DNN architecture and the distribution of the DNN is performed simultaneously, the candidate architecture should have desired accuracies. One research [88] in online federated NAS used an evolutionary NAS algorithm to reduce the computational cost and communication cost. FedNAS [89] implemented differentiable NAS algorithm DARTS which is explained in Section 3.1.

DecNAS [90] is a population-based offline federated NAS that follows a well-defined process. It begins by initializing a global model and assigning resource budgets (latency or computational cost) to clients. Simplified global models are created by neural network pruning and distributed to client groups. Each communication round executes local training, evaluation of test accuracies on validation datasets, and sending results to the server. The server aggregates local models, calculates weighted accuracies, removes poorly performing models, and updates the remaining models with aggregated gradients. This iterative process repeats until convergence. The global model is then replaced and stored, and this continues until convergence is achieved. Finally, federated training is performed on the stored global model. The method incorporates model pruning and client sampling for communication efficiency but acknowledges challenges in accurately representing test accuracies, particularly in non-IID client data scenarios.

FedNAS [89] is a gradient-based method, inspired by DARTS, that employs a global model known as the supernet, consisting of repeated directed acyclic graphs (DAGs) with all candidate operations. Using relaxation tricks [91], a weighted sum of candidate operations becomes differentiable, facilitating direct updates of architecture parameters via gradient descent. The process involves server initialization, clients downloading and locally updating the supernet, and iterative parameter uploading for weighted averaging by the server until convergence.

RT-FedEvoNAS [88] is a real-time multi-objective evolutionary approach that reduces additional communication costs and computational resource overhead while

preserving model performance in the optimization process. The proposed method employs a double-sampling strategy, where submodels are sampled from a global model shared by all individuals within a population, and participating clients are sampled for submodel training. This approach enables the integration of one generation of evolutionary optimization within a single communication round, resulting in a significant reduction in both communication and computation costs.

FedorAS [92] conducts NAS in a federated and resource-aware way, considering heterogeneous devices and data. Clients are clustered based on capabilities, and a supernet is designed as both a search space and a weight-sharing system. The supernet is partially shared with clients, considering their computational and bandwidth capacities. The authors utilize resource-aware one-shot path sampling for lightweight NAS on devices, optimizing neural networks for resource efficiency. The federated training of the supernetwork produces pre-trained networks, reducing the need for on-device training rounds.

4. SYSTEM DESIGN

4.1. Network Emulated Federated Neural Architecture Search

In this thesis research, we present a comprehensive design framework for a Federated Neural Network Pruning system featuring an emulated network environment. The system is based on the fundamentals of Federated Learning, in terms of decentralized DNN model training with local datasets. The neural network pruning algorithm focuses on pruning filters in each layer of CNN iteratively, aimed at reducing the neural network's size and computation cost while maintaining its performance. The federated learning infrastructure is delicately built, comprising server and client components. An emulated network setup is incorporated to simulate federated learning processes on a network, with the benefits of testing and debugging on containers. The neural network pruning algorithm is integrated smoothly into the federated learning process, with consideration for dynamic conditions in the network. Evaluation metrics are defined to address accuracy, multiply-accumulate operations (MAC), inference latency, communication cost, and communication overhead. Multiple testing and validation are conducted to ensure the efficacy of the system, achieving desired model compression and optimizing the speed of communication rounds. This design represents an open-source experiment framework for federated neural network pruning.

4.2. Neural Network Pruning Algorithm

As the neural network pruning algorithm, NetAdapt [28] is utilized. NetAdapt generates a rule-based pruned architecture iteratively and each iteration proposes multiple DNN candidates and selects the best one from them. The original algorithm of NetAdapt uses multiple GPUs for fine-tuning each candidate DNN. Specifically, to finish an iteration and select the best-performing DNN, the algorithm creates subprocesses for each candidate DNN and assigns one GPU for each of them. Therefore, the NetAdapt algorithm is suitable for utilization in the federated learning system.

The NetAdapt algorithm prunes DNN architecture by its block, the term "block" typically refers to a group of layers or a set of operations that are combined to form a building block of the overall CNN architecture. In our case, we have utilized AlexNet [93] architecture in our pruning process. There are 2 types of blocks in AlexNet architecture, blocks with convolutional layers and blocks with fully connected layers(FC). There are 7 blocks in AlexNet architecture therefore in each iteration NetAdapt algorithm proposes 7 different pruned DNN candidates. A sample for the first iteration is illustrated in Figure 4.1.

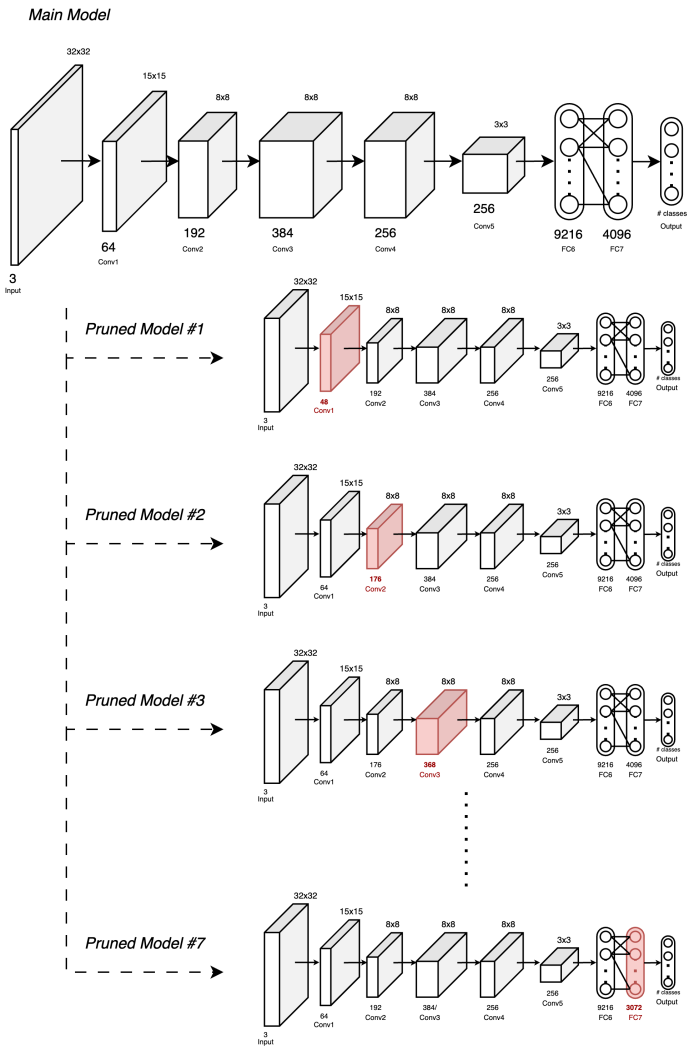


Figure 4.1. Pruning of AlexNet architecture.

Along the iterations, the predefined constraint is reduced gradually. An iteration starts with calculating the resource constraint. The resource of the best model of the iteration defines the resource constraint of the next iteration. As described in Section 2, a resource reduction schedule is enabled, and parameters that define the constraint formulated as

$$\mathcal{R}_{target} = \mathcal{R}_{current} - IRR * (RRD^{i-1}), \tag{4.1}$$

where i is the number of iterations, $\mathcal{R}_{target}$ and $\mathcal{R}_{current}$ are the target resource of the next iteration and current iteration of the pruned best model respectively. IRR is the initial resource reduction and we have set it as $5x10^{-2}$. RDD is the resource reduction decay that is set as 0.96.

4.3. Federated Learning Framework

We have implemented federated learning to the fine-tuning step of the pruned model in each iteration. The server executes the pruning algorithm and generates pruned candidate DNNs for fine-tuning, then distributes those DNNs to client devices for fine-tuning with local data.

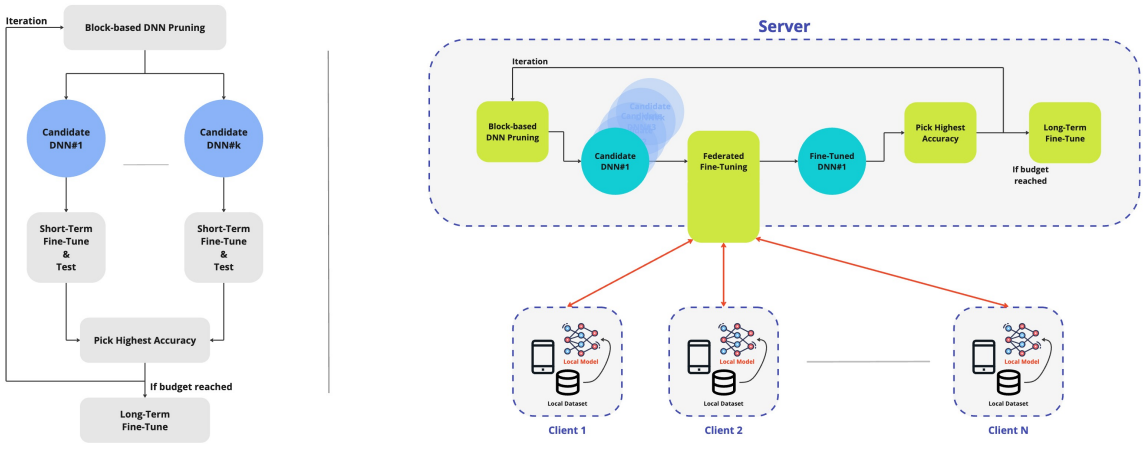


Figure 4.2. Original pruning pipeline with centralized fine-tuning and pruning pipeline with federated fine-tuning.

Each pruned candidate DNN requires a fine-tuning process and a federated fine-tuning strategy can be developed. The straightforward approach is sequentially fine-tuning each model in federated settings. In Section 5 other approaches have been discussed. The implementation of federated fine-tuning is illustrated in Figure 4.2.

4.3.1. Flower FL Framework

To perform federated learning with server and client communication, we have used Flower FL Framework [94]. Flower is a user-friendly FL framework that enables the creation of servers and clients practically for a federated learning pipeline and also provides scalable and secure communication with gRPC protocol between servers and clients. The framework also provides built-in weight aggregation algorithms to implement and utilize on the server side.

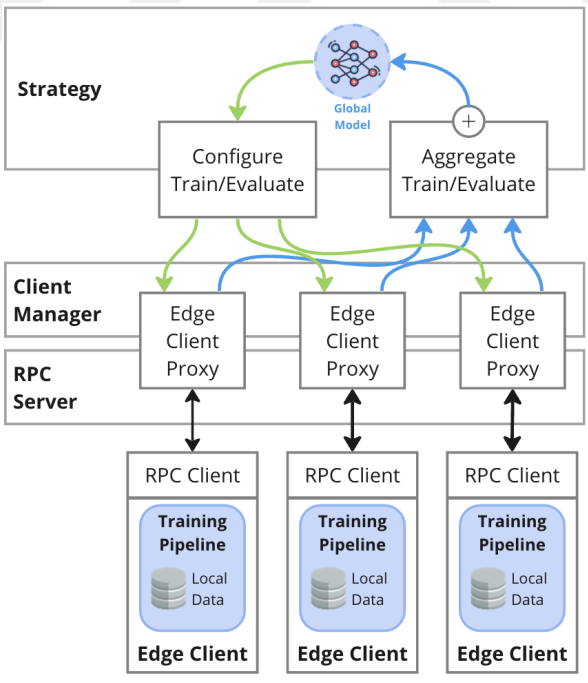


Figure 4.3. Flower core framework architecture with edge clients.

Moreover, it is possible in the framework to customize the federated learning pipeline. The transferred data in the communication can be customized and useful information about the federated learning process or the algorithm can be shared with

the clients. Also, the behavior of the clients in training and evaluation steps can be customized due to the requirements of the whole experiment. In the Flower framework, a strategy is the central component that orchestrates the federated learning process. It encapsulates the core logic that dictates how the learning process happens, making it a crucial element for customizing and controlling various aspects of a federated learning system as shown in Figure 4.3.

4.3.2. Modification in Flower FL Framework

In the Flower framework, custom configurations have been implemented to enhance communication during the federated learning process and adapt federated learning in the neural network pruning scenario.

Traditionally, servers typically send only global model weights to the clients for training and updating. However, our iterative neural network pruning algorithm generates different DNN architectures. Therefore, the server has been configured to transmit not only the model weights but also the entire model architecture. This is achieved by serializing the PyTorch model using TorchScript. By serializing the model architecture, the server is now capable of sending a comprehensive representation of the neural network structure to the clients, enabling a more detailed exchange of information.

On the other hand, clients have also been configured to deserialize the received architecture information in bytes. To this end, since the federated learning is conducted in several rounds, the server only sends the weights and architecture together in the first round to inform the clients about the architecture they are working on.

This approach also can be advantageous in various scenarios where the server requires access to the complete model architecture for tasks such as dynamic architecture-specific changes during the federated learning communication rounds.

4.3.3. Federated Metric Aggregation

In FL, the aggregation of performance metrics, such as accuracy, combines the locally computed accuracy values from clients to obtain a global or federated accuracy measure. This process is crucial for evaluating the overall performance of the global model across the clients where there is not any shared validation data. The federated accuracy is defined where n_i is the total number of training samples if client i, as

$$\text{Aggregated Accuracy} = \frac{\sum_{i=1}^{N} n_i \cdot Acc_i}{\sum_{i=1}^{N} n_i}. \tag{4.2}$$

4.4. Network Emulator

One of the distinctive approaches in our experiments of this research is the network emulator implementation to federated neural architecture pruning. Federated learning cannot be thought of without network conditions and architectures. Network emulation has various advantages for our experiments in neural network pruning with federated settings.

A network emulator allows you to simulate real-world network conditions, including varying bandwidth, latency, and packet loss. This ensures that the federated learning system performance is tested under conditions that the actual deployment environment.

Federated learning usually involves resource-constrained devices. Network emulators enable testing management of resources such as bandwidth and computational power by the system in different network conditions.

4.4.1. CORE Network Emulator

CORE, the Common Open Research Emulator [95], stands out as a powerful network emulation tool for research, development, and education. As an emulator, different from a simulation where abstract models are used, CORE setups a represen-

tation of a real network that behaves in real time. Its user-friendly interface allows us to design custom network topologies. On the other hand Python module further enables automation of setting up various network topologies with different sclaes for extensive testing securities.

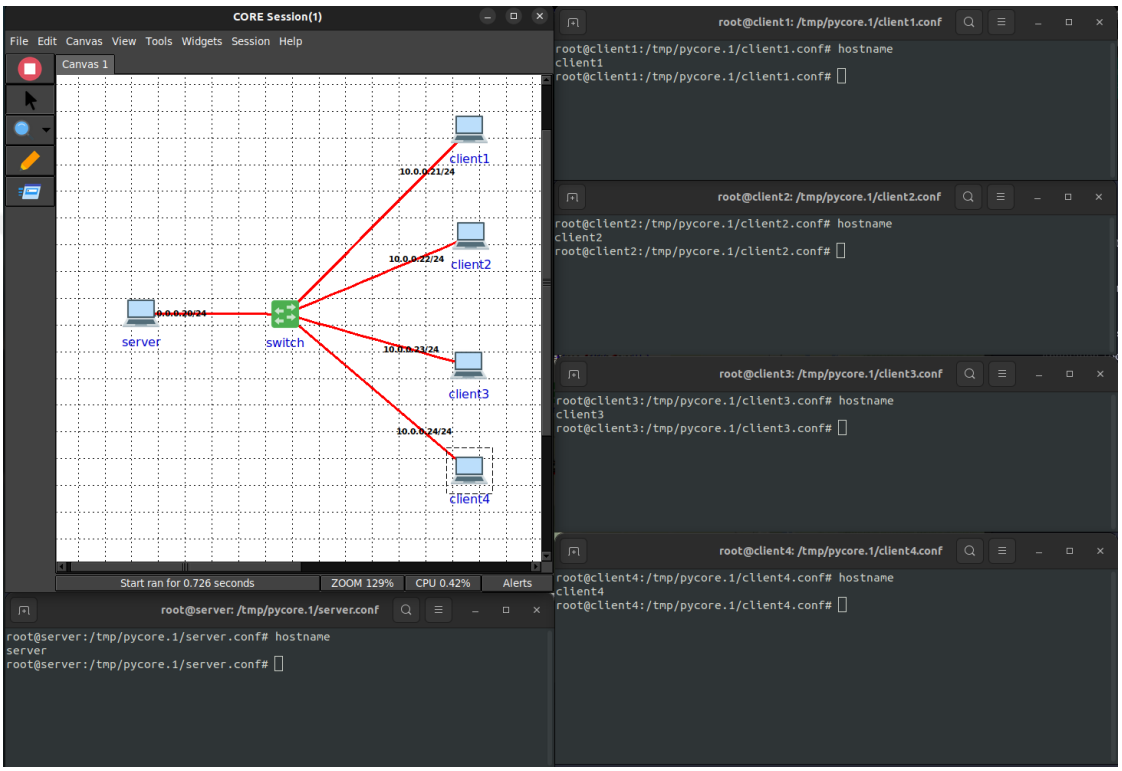


Figure 4.4. User interface of CORE.

CORE does offer a user interface as shown in Figure 4.4 as well as a Python API which enables users to interact with the emulator programmatically. This API can be used to automate emulator session setup and create custom nodes. The API allows Python scripts to manipulate network elements, configure nodes, and control the behavior of the emulated network. It is possible to configure link properties such as bandwidth, delay, duplicate, jitter, and loss between nodes during runtime.

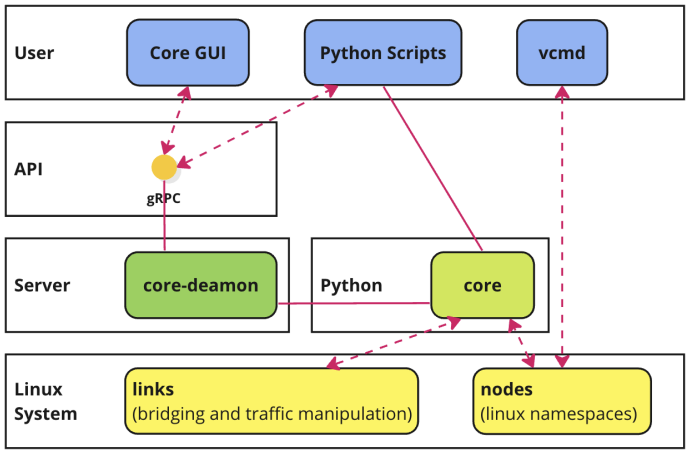


Figure 4.5. Architecture of CORE.

The core-daemon is a crucial part of the CORE framework, as shown in Figure 4.5, which helps simulate networks for research. It manages devices (nodes) and connections (links) in a way that emulates real networks. Using Linux namespaces, it creates these devices and connections. It also controls how data moves through these connections to make the simulation realistic. Additionally, the core-daemon has a gRPC API, making it easy to control and interact with the simulated networks in the CORE emulator.

In our research using the CORE framework, we have selectively employed various network topology components to create targeted and realistic scenarios. Particularly, we have focused on utilizing custom nodes and switches as the primary components within our federated learning network configurations. Custom nodes afford us the task specific network devices for our research requirements. Moreover, we have integrated Docker containers within these custom nodes, leveraging containerization for the deployment of flower servers and clients. This approach allows us to develop a seamless incorporation of the containerized Flower FL framework within the broader network topology.

4.4.2. Configurations in CORE

In our experimental setup, we have incorporated the Flower FL framework, specifically utilizing servers and clients encapsulated within Docker containers. For fine-

tuning DNNs on the client side, we have imposed strict resource constraints on the client containers by allocating 8 CPU cores and 6GB of memory. This restriction aims to simulate realistic computing conditions and assess the pruned DNN performance within resource-constrained environments. Our network configuration within CORE encompasses a total of 56 Flower clients. Each client does not contribute to the distributed fine-tuning process every time, the client selection and grouping are discussed in Section 5.

By leveraging the Python API provided by CORE, we automated the creation of a CORE session that emulates a server container and multiple client containers interconnected in a network topology. In our topology design, the server is connected to a switch, and all client containers are linked to this switch. We have defined 3 tiers of bandwidth conditions as low, medium, and high; and a tier is assigned to each link between the clients and the switch. A dynamic adjustment mechanism is introduced to the network configurations wherein the bandwidth for each link in the network is modified every 5 minutes within the predefined limits of the assigned tier.

We purposely adjusted the bandwidth levels to imitate real-world scenarios where bandwidth fluctuation occurs. This variation allows us to assess how effectively the federated learning model behaves in different bandwidth situations.

4.5. Proposed Client Grouping Algorithm: NetDOG

We have proposed a client grouping algorithm, named **Net**work and **D**istribution **O**ptimized Client **G**rouping (NetDOG), is designed to enhance the efficiency of model training in federated learning settings by considering both network bandwidth conditions and non-IID class distribution of clients. The objective behind this algorithm is strategically assigning clients to different DNN models (in the context of neural network pruning) based on network bandwidth and the class distributions of respective datasets.

Algorithm Network and Class Distribution Optimized Client Grouping (NetDOG)

Input: MAX_ITER: Maximum number of iterations. N: Number of groups. $COEFF$: Coefficient for client transfer threshold.

Output: G: Group numbers of each client.

$G \leftarrow$ Assign groups to clients by simple network-aware client grouping
$sequence_of_groups \leftarrow [1, 2, 3, ..., N]$
for iter $= 1$ **to** MAX_ITER **do**
  $sequence_of_groups \leftarrow$ shift the sequence
  $noOperationsDone \leftarrow 0$
  for K **in** $sequence_of_groups$ **do**
    $currentAverageEMD \leftarrow \frac{1}{N}\sum_{n=1}^{N} \text{EMD}(Dist_{G_n}, Dist_G)$
    $neighbourClients \leftarrow \{C_{K-w}, ..., C_{K+w}\}$
    $candidateOperations = \{\}$
    for c_i **in** C_K **do**
      for c_j **in** $neighbourClients$ **do**
        $G^{temp} \leftarrow transfer(c_i, c_j)$
        if $bw_i = bw_j$ **then** coeff $= 1$
        else coeff $=$ COEFF
        if $EMD(G_{g_i}^{temp}) < EMD(G_{g_i}) \times coeff$ **then**
          $newAverageEMD \leftarrow \frac{1}{N}\sum_{n=1}^{N} \text{EMD}(Dist_{G_n^{temp}}, Dist_G)$
          Append operation(c_i,c_j,$newAverageEMD$) to $candidateOperations$
    c_i, c_j, $averageEMD \leftarrow$ select operation with minimum $averageEMD$
    if averageEMD $>$ currentAverageEMD **then**
      $G \leftarrow transfer(c_i, c_j)$
      $noOperationsDone++$
  if $noOperationsDone = 0$ **then** break

Figure 4.6. NetDOG algorithm.

The algorithm optimizes communication efficiency by assigning clients with lower network bandwidth to models with smaller sizes. Especially assigning to a group that desired to train a specific model. This strategic assignment aims to alleviate potential communication bottlenecks by shifting the communication overhead to smaller models, in order to reduce communication time and optimize the overall federated learning process.

Furthermore, the NetDOG takes into account the non-IID datasets of clients with varying class distributions. To address this challenge, the algorithm employs an iterative process that optimizes the class distribution within each client group. This ensures that each global model that is trained federally receives a diverse and balanced set of data representations, providing a robust and generalized global model. The NetDOG iteratively transfers clients between groups based on the EMD values of the groups, aiming to achieve a balanced and representative distribution of classes.

The pseudocode of NetDOG is given in Figure 4.6. The algorithm initiates by assigning clients to groups using a simple network-aware approach. It then iteratively refines these assignments over a specified number of iterations, *MAX_ITER*, to achieve enhanced optimization. It strategically evaluates these transfers based on their impact on the EMD value of the groups after the transfer. The algorithm expects an improvement as the coefficient in average EMD value if the clients in a transfer pair are in different bandwidth tiers, in other cases expectation is just an improvement in average EMD.

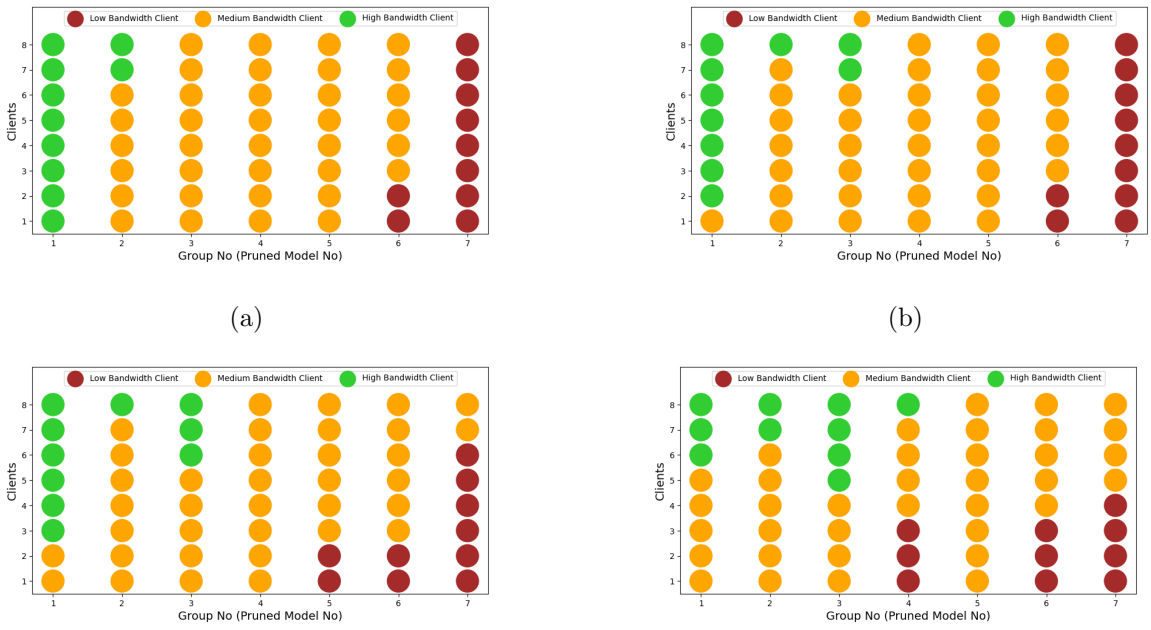


Figure 4.7. Groups generated by NetDOG in different COEFF settings such as (a) 0.5, (b) 0.7, (c) 0.9 and (d) 1.0.

The algorithm has individual clients c_i assigned to specific groups g_i with bandwidth tiers bw_i. It holds on the entire set of clients C and its subsets C_j based on group number j. Similarly, it considers the complete set of groups G and group subsets G_j. Class distributions are captured for individual clients $Dist_{c_i}$, their assigned groups $Dist_{g_i}$, the entire dataset $Dist_G$, and group subsets $Dist_{G_j}$. Temporary variables, g_{temp} and G_{temp}, help in exploring client transfers. Algorithm behavior is controlled by the maximum iteration count MAX_ITER, transfer threshold coefficient $COEFF$, total number of groups N, and window length w used in calculations.

Table 4.1. Comparison of EMD values of Different COEFF values.

COEFF	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Average**
0.5	0.34	0.25	0.33	0.23	0.19	0.20	0.60	0.31
0.7	0.34	0.32	0.28	0.31	0.23	0.25	0.57	0.33
0.9	0.33	0.17	0.22	0.24	0.34	0.21	0.26	0.26
1.0	0.21	0.27	0.26	0.22	0.31	0.22	0.12	0.23

Selection of the *COEFF* parameter is crucial while allowing transfers to happen. Therefore, it prohibits the possible outcomes of the algorithm in terms of the groups and their distributions. We have conducted a preliminary test to select an appropriate *COEFF* value by comparing the resulting EMD values of groups to achieve better accuracy and the client groups to observe the low bandwidth bottlenecks. We have selected *COEFF* as 0.5, 0.7, 0.9 and 1.0. The resulting groups have been given in Figure 4.7(a), Figure 4.7(b), Figure 4.7(c) and Figure 4.7(d). For the values lower than 0.7, the transfers have been highly restricted and nearly no transfers happened between different bandwidth tiers. 1.0 refers to no restriction and every possible transfer that reduces the EMD average happens. On the other hand, the lowest value EMD is provided with 0.1, Table 4.1. However, the best setting that balances the trade-off between the bottleneck that generates communication latency and class distribution of groups is provided with setting *COEFF* to 0.9.

4.6. Overview of NAFNAS

The designed system consists of the fusion of the NetAdapt neural network pruning algorithm, Flower FL framework, and CORE Emulator for comprehensive network simulation. The neural network pruning algorithm generates pruned candidate DNNs and utilizes a Flower Server to conduct federated fine-tuning. The Flower Framework is built upon the CORE network emulator in order to enable an enhanced experiment environment in network topologies.

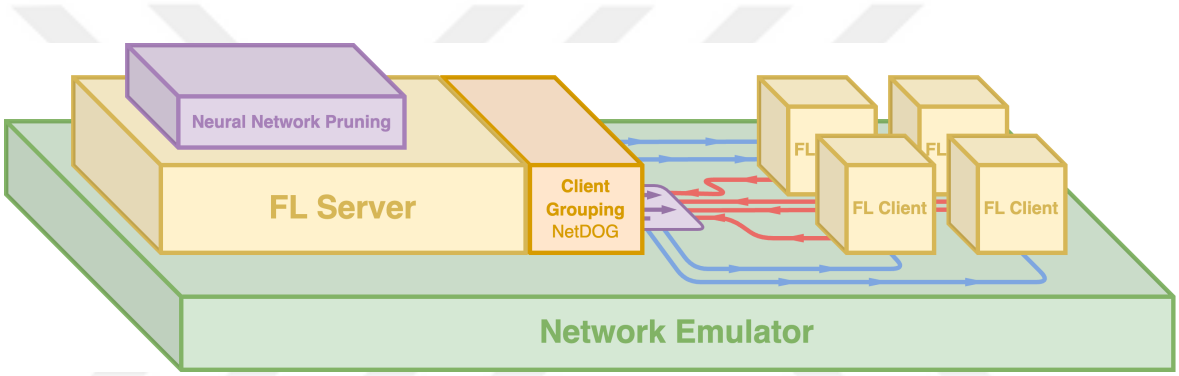


Figure 4.8. Overall NetDOG System Architecture.

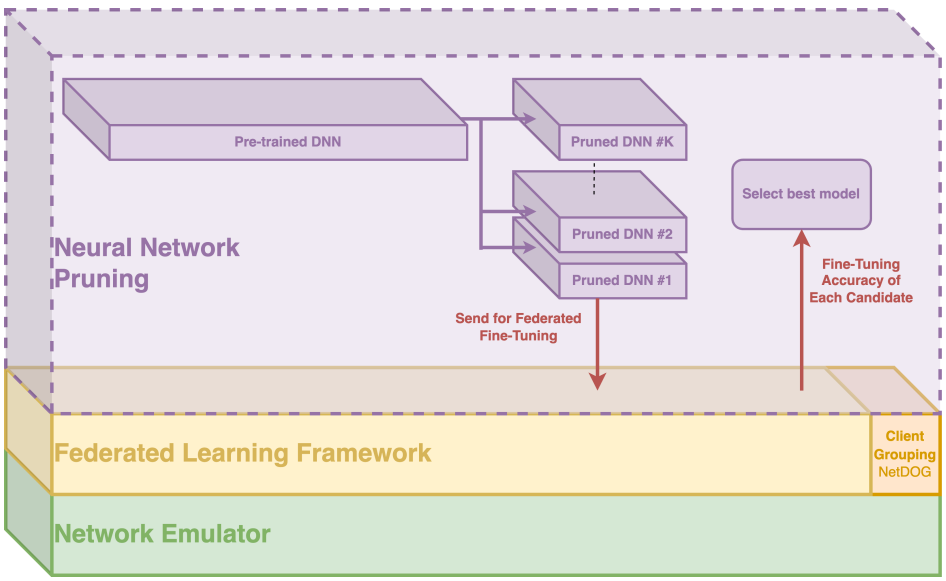


Figure 4.9. Neural Network Pruning in The System.

In the NAFNAS, whose overall system is illustrated in Figure 4.8, the network pruning algorithm generates multiple pruned candidate DNNs by pruning the CNN blocks individually. In each iteration, the best performing DNN from candidates is selected by fine-tuning all candidates in federated settings. For that purpose, each candidate is distributed by a Flower Server to enable federated fine-tuning. After DNNs are fine-tuned, their accuracies are collected for selecting the best-performing model. The relation between NetAdapt and Flower Server is illustrated in Figure 4.9.

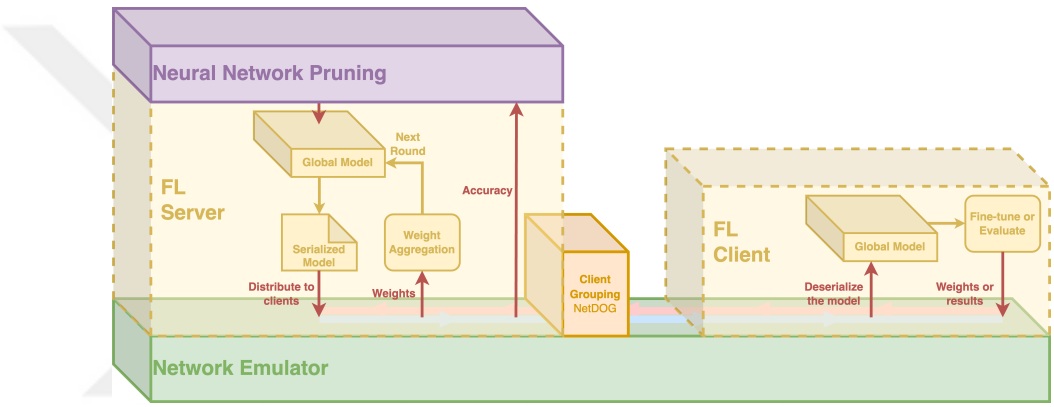


Figure 4.10. FL Framework in The System.

Before continuing with network emulator, our implementation of client grouping algorithm NetDOG was placed with Flower Server in order to guide the server to select clients to distribute the DNN models. The algorithm behaves as an intermediate operator that collects bandwidth information from the network and class distributions of clients from the server. Then uses that knowledge to guide the server.

When the Flower Server is utilized for a candidate DNN, the server serializes the global model (candidate model for pruning algorithm) and distributes the global model to clients. The client either fine-tunes or evaluates the received model and sends updated weights or evaluation results to the server. The network topology is built upon the CORE network emulator with dynamic bandwidth settings. The details of federated learning on Flower Framework are shown in Figure 4.10.

5. EXPERIMENTS AND RESULTS

5.1. Experiment Settings

5.1.1. Dataset

CIFAR10 is a widely used dataset in the world of computer vision and machine learning. It contains 60,000 color RGB images, each of them sized 32x32 pixels, in 10 classes such as airplanes, cars, birds, and cats. The dataset's relatively low image resolution and diverse content make it a challenging but valuable resource for developing accurate and adaptable models. CNNs are often utilized to handle the complexities of this dataset, establishing CIFAR-10 as a key benchmark for evaluating and comparing different DL approaches in image recognition.

5.1.2. Model

AlexNet [93] is a deep CNN architecture designed for image classification tasks. The model consists of eight layers, including five CONV layers and three fully connected layers. AlexNet was designed and trained on the ImageNet dataset, with samples of 3x256x256 input size. However, we modified the AlexNet for the Cifar10 dataset whose samples are sized in 3x32x32 by reducing the stride and filter sizes.

5.1.3. Resource Type

MAC (Multiply-Accumulate Operations) is a computational operation that accounts for multiplication and addition in a single step. In CNNs, MAC operations happen in the CONV layers where kernels are applied to input feature maps. Therefore, MAC operations are a critical component for measuring the computational efficiency and effectiveness of deep learning models.

5.1.4. Hardware Settings

We have utilized computational resource-limited containers as edge devices or clients in our experiments. Each client has 8xCPU Cores and 6GB of memory, which is a very limited constraint for a case such as CNN training. All the experiments were conducted on the HPC server of Boğaziçi University.

Table 5.1. Bandwidth tiers of clients.

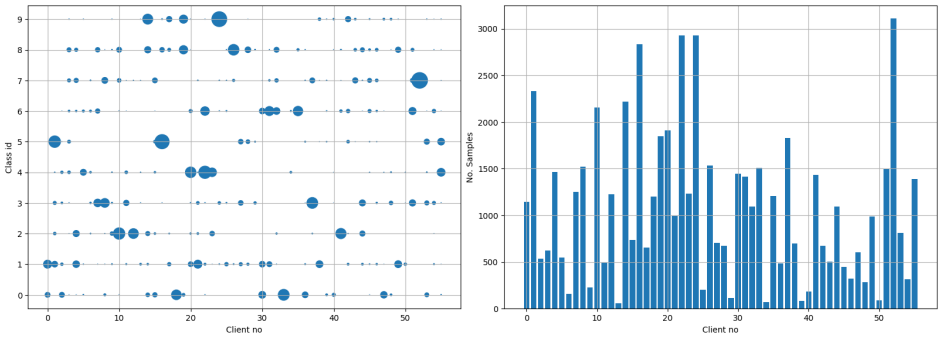
Bandwidth Tier	**Minimum**	**Maximum**	**Step**	**Frequency**	**No. Clients**
Low	2.5 MBps	3.5 MBps	0.5 MBps	5 minutes	10
Medium	8.0 MBps	32.0 MBps	1.0 MBps	5 minutes	36
High	50 MBps	100 MBps	10 MBps	5 minutes	10

5.1.5. Network Configuration

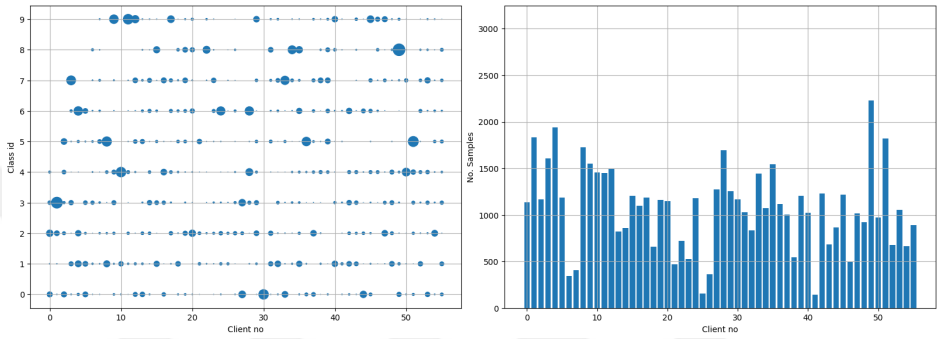
In order to emulate the near real-world scenarios, we have implemented dynamic bandwidth change to the federated learning environment with the help of the network emulator. There are 3 tiers of bandwidth levels in our experiments, as given in Table 5.1, and each tier has an upper and lower limit and step that defines the array of bandwidth values between limits. With the 5 minutes update frequency, the bandwidth of every client is randomly selected from an array.

5.2. Non-IID Dataset Generation of Clients

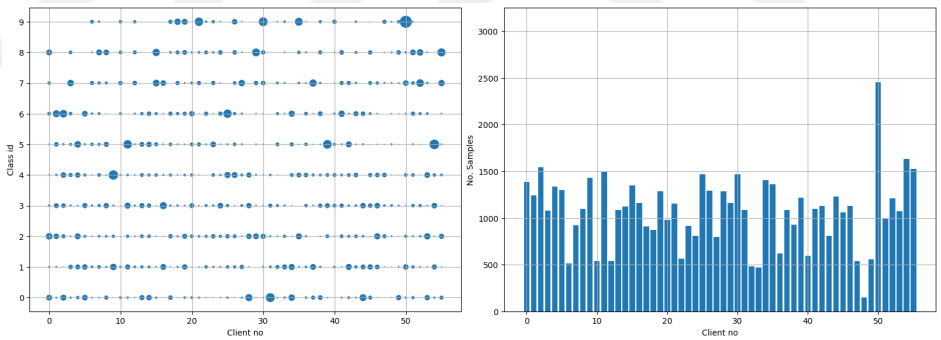
We have aimed to split the Cifar10 dataset to clients in non-IID settings to simulate real-world scenarios in federated learning. Dirichlet distribution is utilized to split the uniformly distributed dataset and assign it to clients in non-IID splits. The level of the non-IID split is controlled by a parameter in the Dirichlet algorithm called α. The controller parameters α are used to be selected between 0 and 100 in order to change the level of non-IID.



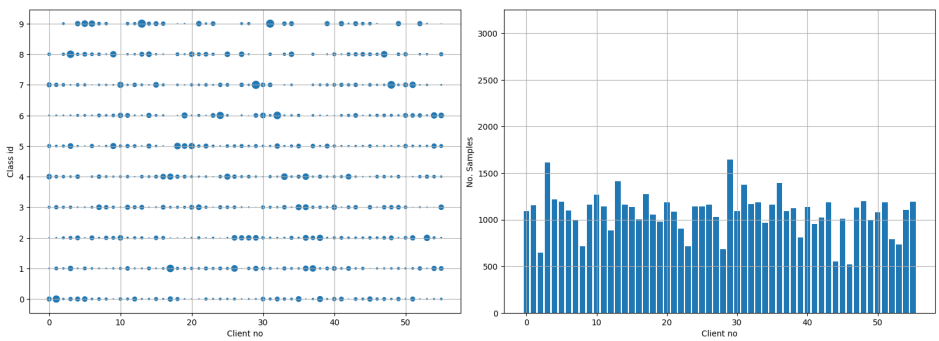
(a)



(b)



(c)



(d)

Figure 5.1. Class distributions and total number of samples of each client where (a) $\alpha = 0.1$, (b) $\alpha = 0.3$, (c) $\alpha = 0.5$ and (d) $\alpha = 1.0$.

In Figure 5.1(a), Figure 5.1(b), Figure 5.1(c) and Figure 5.1(d), we have experimented with the class distributions of each client in different α configurations as [0.1 0.3 0.5 1.0]. With greater values of α, the class distributions of clients become closer to uniform distribution as Cifar10 class distribution. Lower values of α generate more class-skewed distributions and also affect the total number of samples in each client. The total number of clients is important because of the fact that more training samples require more time for training the global model in edge devices. With the $\alpha = 0.1$ the distribution becomes more skewed.

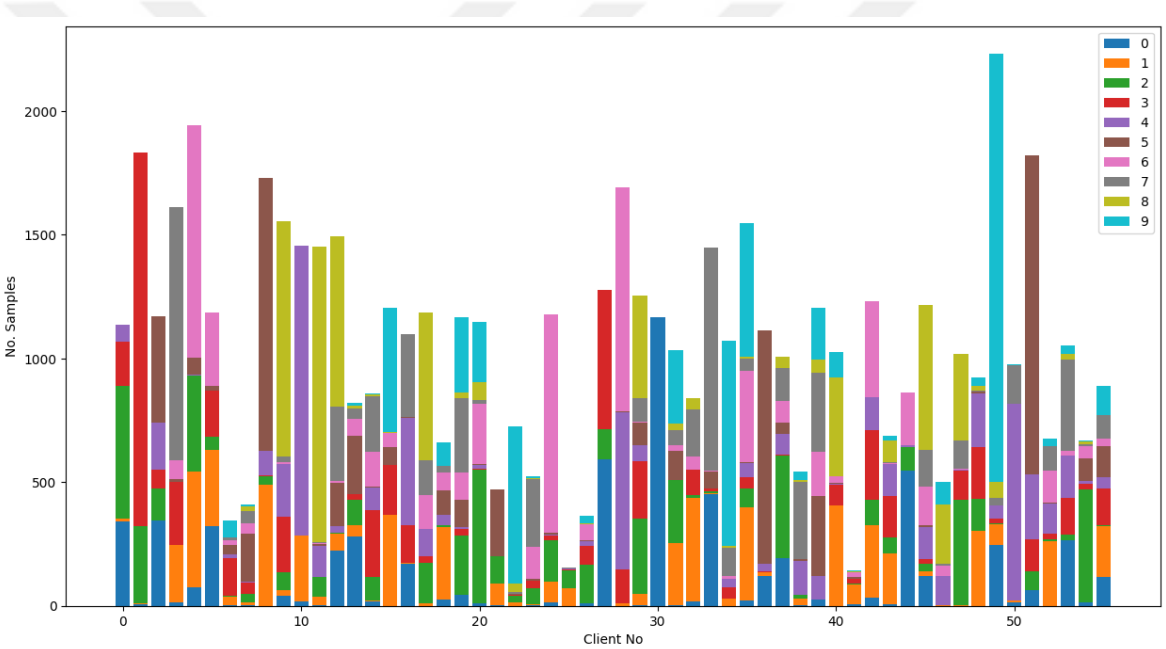


Figure 5.2. Number of samples of each class in each client.

We have set $\alpha = 0.3$ since it generates highly non-IID datasets for clients and even though the number of samples is more balanced than 0.1, clients still vary in terms of the sample sizes. The number of samples of each class in each client is illustrated in Figure 5.2.

5.3. Experiments of Federated Learning Framework

NAFNAS consists of a federated learning framework for training and fine-tuning the DNN models. First of all, we have tested the federated learning framework in

order to achieve a reliable and healthy training process and generate the pre-trained model which is going to be used in neural network pruning. The performance of the pretrained model is crucial in neural network pruning since the pruned models highly rely on the weights of the parent model.

Clients have been grouped randomly for the initial model training and in each communication round of federated learning, one group is selected from all groups sequentially. In the experiments, there were 56 clients and the dataset whose class distribution is uniform was split to those clients in a non-IID manner. For the sake of simplicity, 56 clients are grouped into 7 groups since there are 7 pruned models in every iteration of the neural network pruning algorithm.

AlexNet model is utilized in the federated training process. After one group is selected in a communication round, the model weights are distributed to clients in the selected group. Each client trains the received model for 50 epochs with a Stochastic Gradient Descent optimizer whose learning rate is 10^{-3} and weight decay is 5×10^{-4}. The federated learning process is continued for 300 rounds with different groups of clients as mentioned in the previous paragraph.

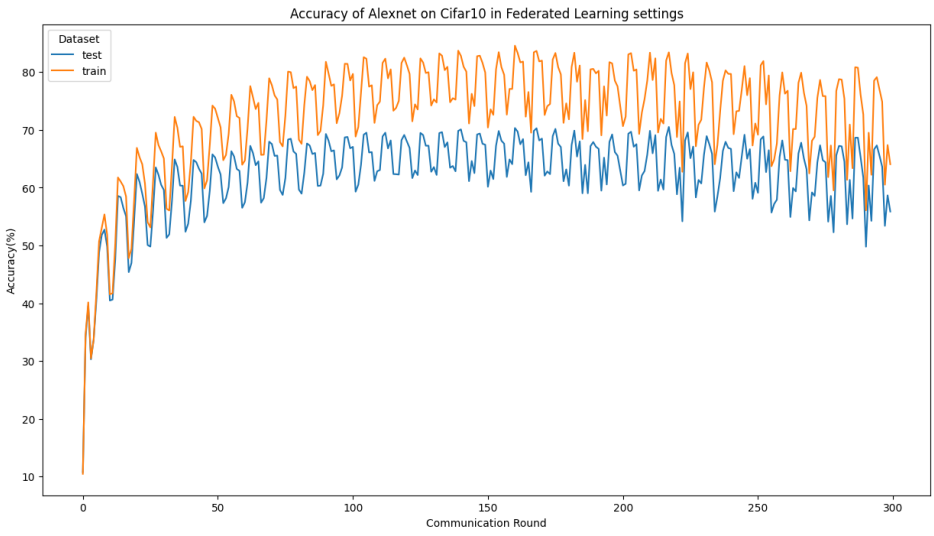


Figure 5.3. Aggregated training and validation accuracies of Alexnet on Cifar10 dataset in federated learning settings.

However, the initial AlexNet model is a relatively huge model for such a task, in the federated learning settings with FedAvg the accuracy drops of the non-IIDness of the local datasets. In each communication round, the train and test accuracies have been calculated by a metric aggregation method. The aggregated test accuracy reached to 55.88%. The performance of the AlexNet model in federated learning is illustrated in Figure 5.3.

5.4. Experiments of Federated Neural Network Pruning

NAFNAS implements the neural network pruning algorithm and federated learning framework in order to preserve data privacy in the pruning process. Since the algorithm requires a pre-trained model to prune and propose new pruned models, we have utilized the AlexNet model that is trained in a federated learning framework.

The neural network pruning algorithm starts pruning by each layer and selects the pruned models whose resource satisfies the constraint. The resource reduction through the iterations is formulated in Equation 4.1. The resource type is selected as MAC (Multiply-ACCumulate operations), therefore the target is reducing the MAC. The $\mathcal{IRR}$ and $\mathcal{RRD}$ parameters were set as 0.025 and 0.96, respectively and the pruning process continued for 15 iterations.

The federated fine-tuning process in the pruning algorithm is conducted with 56 clients but again the clients are grouped into 7 groups where each group is assigned to one pruned model for federated fine-tuning. The groups are randomly created, therefore any network bandwidth and class distribution aspects are not considered. Each client fine-tuned the assigned model for 50 epochs for 20 rounds.

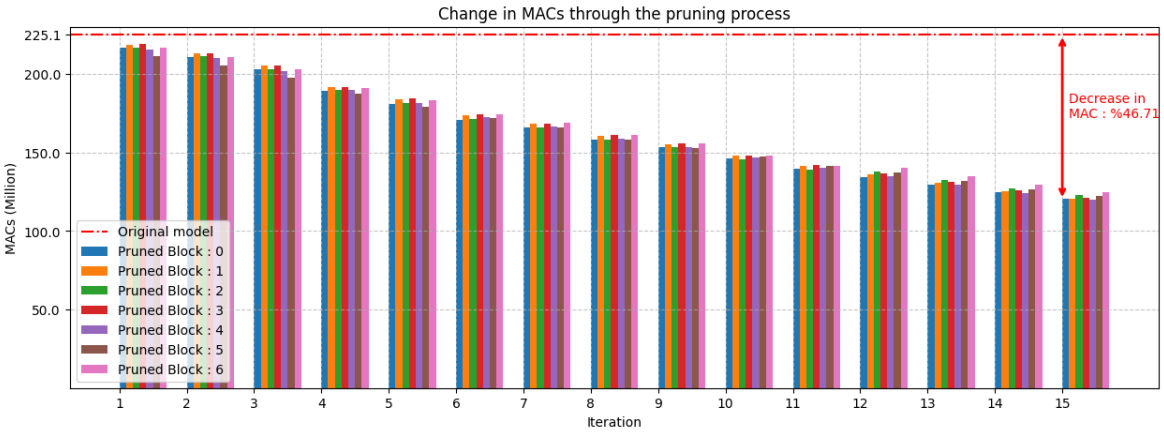


Figure 5.4. Reduction in MACs.

Through 15 iterations in the neural network pruning, the resource (MAC) is reduced linearly where in each iteration the resource is reduced by an average of 4.27 percent. As shown in Figure 5.4, the overall reduction in the resource is 46.71% at the end of the 15^{nd} iteration. Especially, the resource reduction of the 0^{th} and 2^{nd} blocks are higher in each iteration however the selection is not based on the lowest resource, the selection of the best performing model is the pruned model that satisfies the resource constraint with highest accuracy. Therefore, it can be interpreted as the models where 0^{th} and 2^{nd} blocks pruned are not selected as the best model in the iteration since the models where these blocks pruned usually generate lower resource values in most of the iterations.

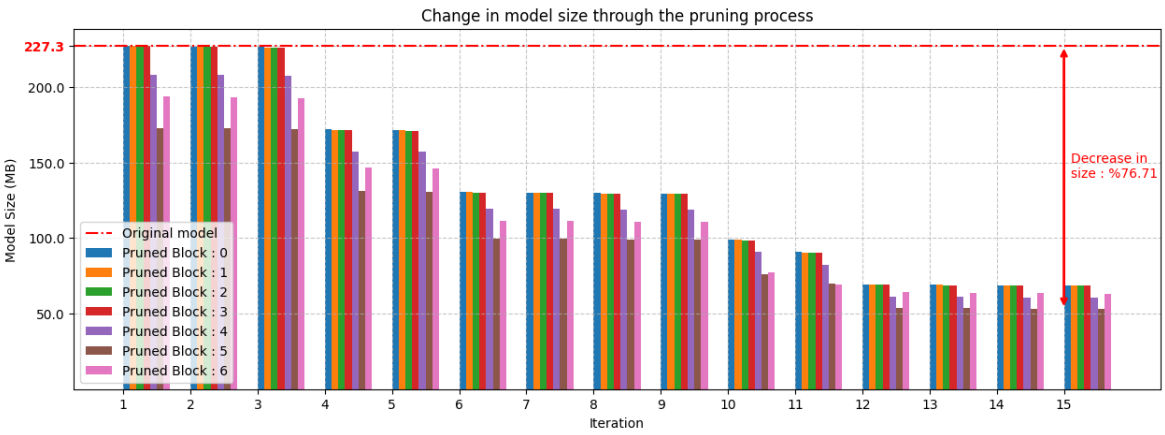


Figure 5.5. Reduction in model size (MBytes).

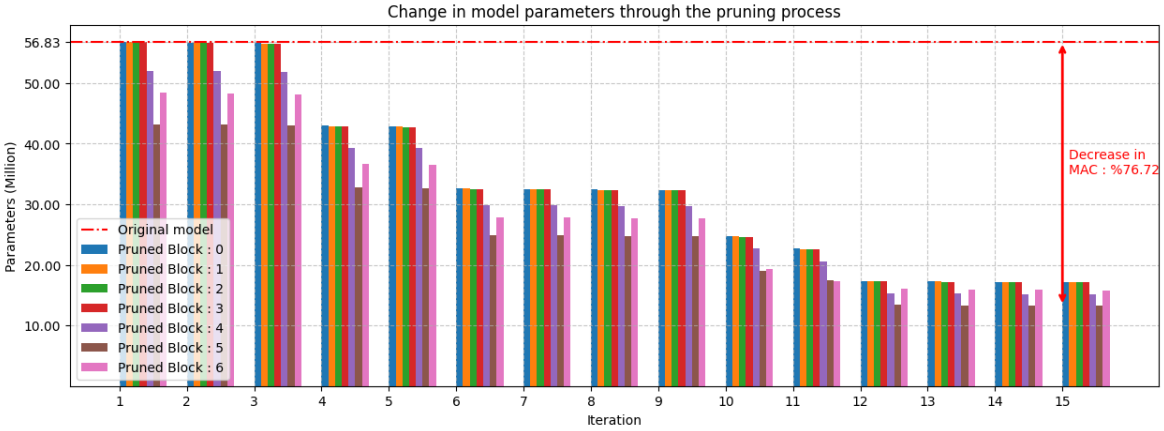


Figure 5.6. Reduction in model parameters.

On the other hand, the number of model parameters and the model size in terms of megabytes behave differently and reduce more radically. When Figures 5.5 and 5.6 are investigated, the total decrease in model size is 76.7% where it is greater than the reduction in MACs. This is because of that there are convolutional and fully connected layers in the AlexNet model which has different effects on MAC and model size, and the neural network pruning algorithm decreases the number of filters in convolutional layers or the number of neurons in fully connected layers. The model size highly depends on the number of trainable parameters while the MAC is based on the number of operations. As shown in Figure 5.5, when the last two layers, that are fully connected layers, are pruned, the model size reduces significantly, and if the best model is the model whose 5^{th} layer is pruned, the model size drops and therefore the resource constraint in the next iteration drops significantly as well (3^{rd} to 4^{th} iteration in Figure 5.5).

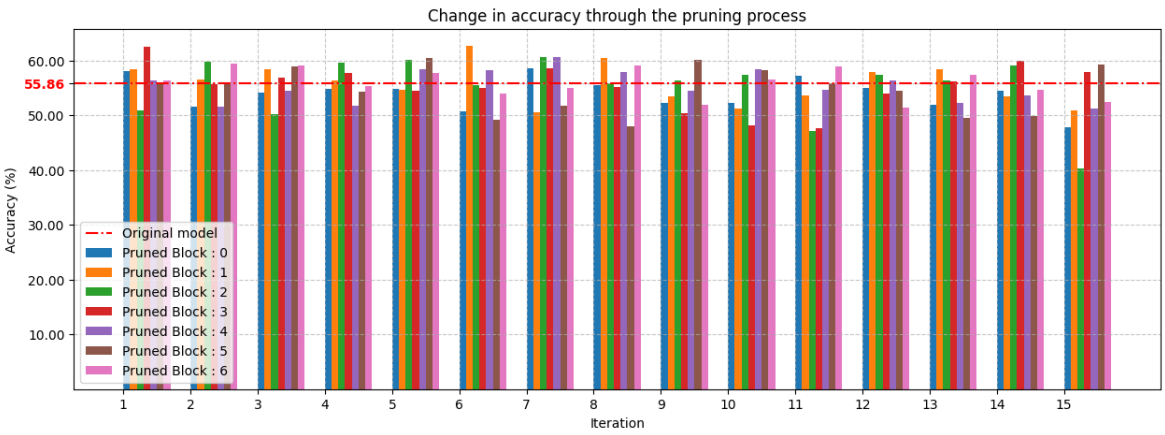
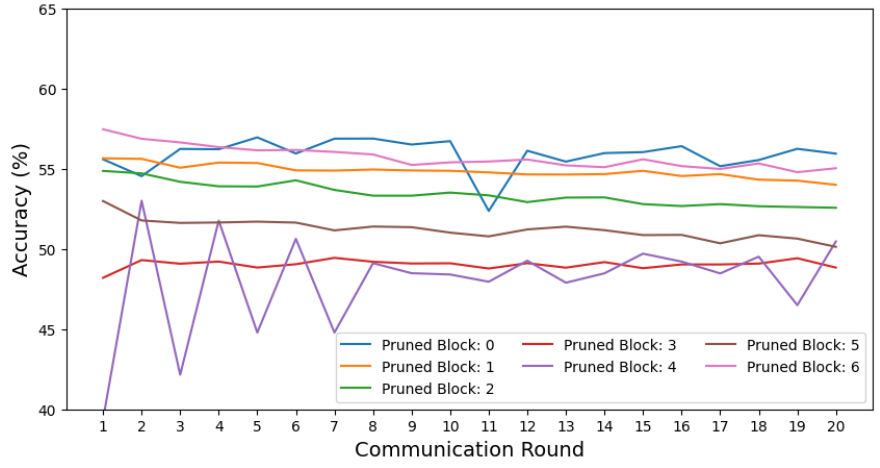
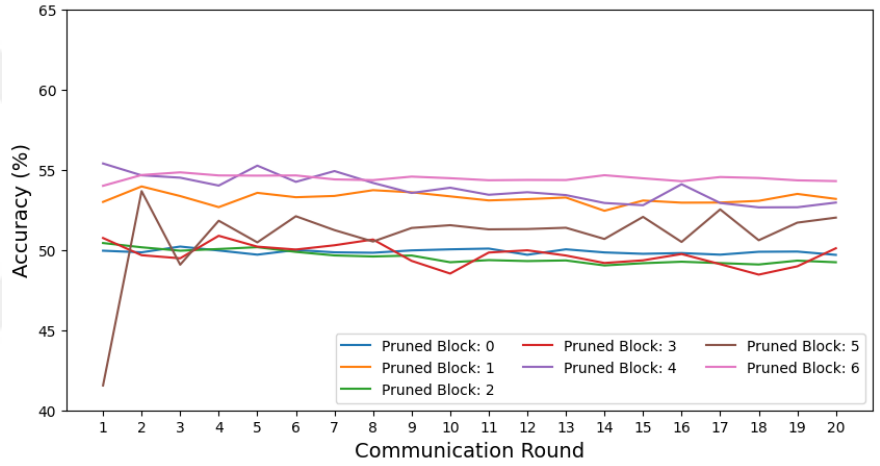


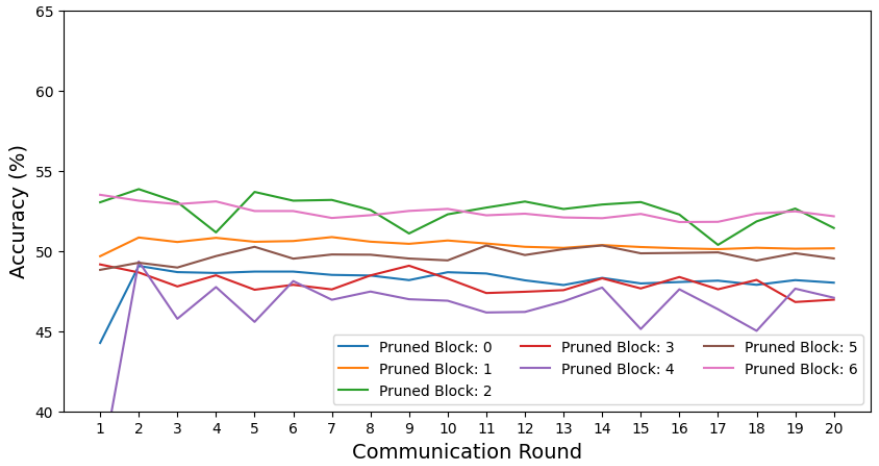
Figure 5.7. Change in accuracy.



(a)



(b)



(c)

Figure 5.8. Federated fine-tuning of each model generated in (a) iteration 1 (b) iteration 6 and (c) iteration 15.

MAC and model size are reduced by 46.71% and 76.72% respectively. As shown in Figure 5.7, the accuracy of pruned models on the Cifar10 dataset is preserved through the iterations while pruning the model. These results mean that the pruned and federally fine-tuned versions of the AlexNet model are as complex enough to classify the samples as the pre-trained version of the model.

Figure 5.8(a), Figure 5.8(b) and Figure 5.8(c) shows the individual pruned model performances through communication rounds in different iterations of pruning. At the first iterations, the accuracy is preserved through the communications rounds therefore the rounds may be reduced, for optimization. After the iterations are continued and the models become more pruned, initial performances of the pruned models reduce therefore federated fine-tuning becomes more beneficial in order to estimate the final performances of the pruned models such as in the 12^{th} iteration.

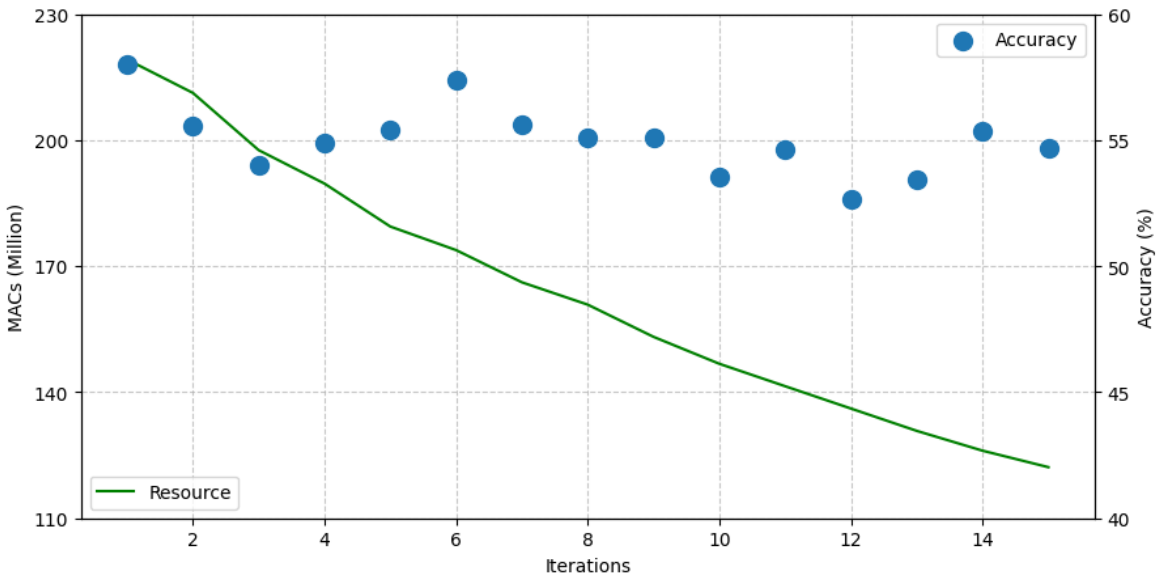


Figure 5.9. MAC and aggregated accuracies of selected models in each iteration.

In the federated neural network pruning, federated learning is conducted in the fine-tuning phase of the pruned models. Through the iterations, it is clear to say that the pruning algorithm reduced the model size and MAC successfully. The reduction in the model size is important from the communication cost perspective since we have utilized federated learning in an iterative process. Figure 5.9 summarizes the experiments

by comparing MACs and accuracies of the best-performing models of each iteration. The accuracy is preserved nearly as the model is not pruned at all, however, the reason is that the AlexNet model is capable of easily handling the Cifar10 dataset by the parameter size and the complexity of the model.

5.5. Experiments of Network-Aware Federated Learning

In NAFNAS, we have implemented network emulation tools in order to observe behaviors of federated neural network pruning from the network aspect, especially bandwidth. The strategic assignment of clients with low bandwidth to model groups holds significant importance. Low bandwidth clients, characterized by limited data transfer speed, become bottlenecks in the efficiency of the communication process. When one of these clients is placed in a group, their constrained model weight exchange capabilities slow down the overall convergence and duration of the federated learning algorithm. In detail, when a model is being trained in federated learning settings the initial global model is distributed to participant clients, in continuation aggregation of local models to build the new global model, the server has to wait for model updates from each client. Even if a single client becomes late, the overall duration of federated learning increases.

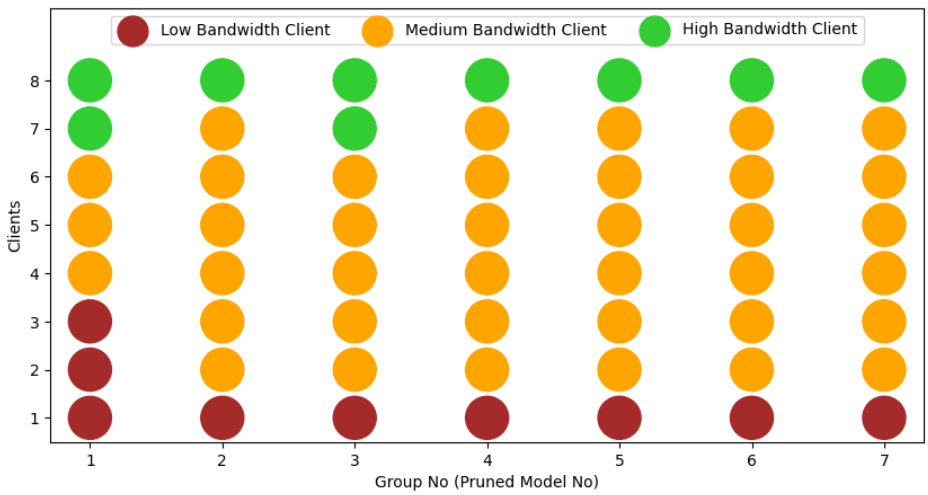


Figure 5.10. Groups generated by basic grouping.

Table 5.2. EMD values of basic groups.

Group No.	**1**	**2**	**3**	**4**	**5**	**6**	**7**
EMD	0.58	0.57	0.62	0.49	0.42	0.41	0.67

In the *basic grouping*, clients have been placed randomly. Therefore, to ensure the statistical significance we have conducted multiple experiments. But in our network-aware experiments, the worst-case scenario is that clients are grouped where all the groups have at least one client from each bandwidth tier. The groups and distribution of the clients are illustrated in Figure 5.10. Class distributions of the groups are important in the federated fine-tuning of the models assigned to the specific group. As described in Section 2.3.2, EMD is utilized to quantify the level of non-IID. In Table 5.2, EMD values of basic groups are given.

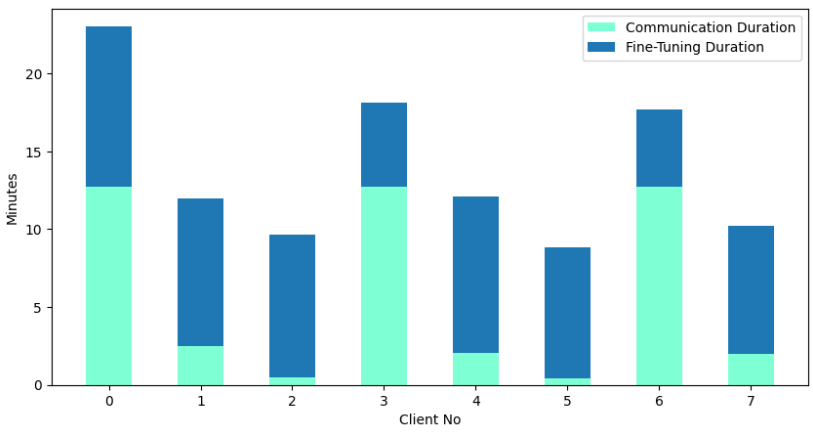
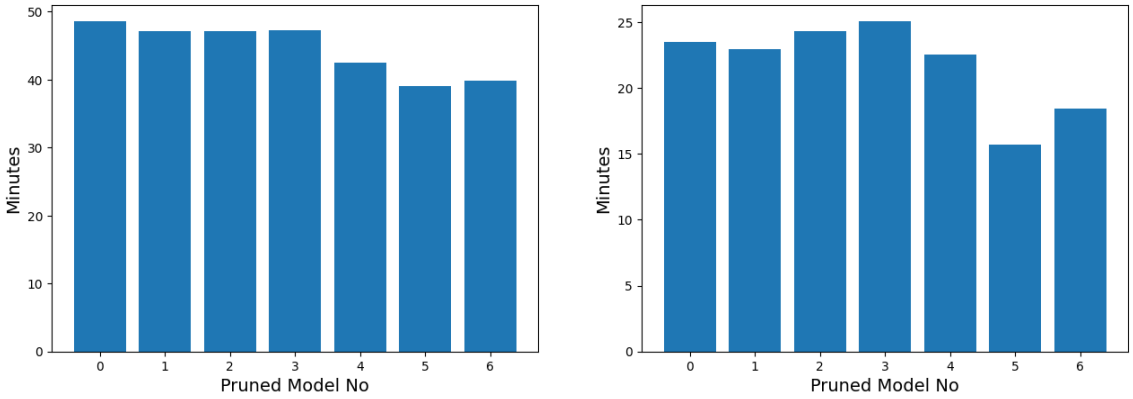


Figure 5.11. Communication and fine-tuning durations for clients in a group assigned for single model pruned in 1st iteration.

Since there are low bandwidth clients in each group, every group has a bottleneck in the communication duration. The Figure 5.11 shows that low bandwidth clients such as 0th, 3rd and 6th require more communication time. The client devices have limited computational resources in terms of CPU and memory, as described in Section 5.1.4, therefore the fine-tuning duration would be significantly longer than fine-tuning on a server machine with GPU acceleration. When compared to fine-tuning duration, communication duration becomes a crucial bottleneck for a round of fine-tuning.



(a) (b)

Figure 5.12. Overall durations of one communication round in (a) 1^{st} and (b) 10^{th} iterations for pruned models with basic grouping.

The duration of one round of communication highly depends on both the bandwidth bottleneck in a group and the sizes of the pruned model that is assigned to the group. In Figure 5.12(a) and Figure 5.12(b), the overall federated fine-tuning duration for one communication round is visualized for 1^{st} and 10^{th} iterations of neural network pruning. Because the model sizes are reduced through the iterations the round duration is also reduced as expected and also the round duration for the models whose 5^{th} and 6^{th} blocks, fully-connected layers, is relatively shorter. That means assigning low-bandwidth clients to 5^{th} and 6^{th} models would reduce the overall duration of the iteration.

With considering the network bandwidths of the clients, clients can be grouped for the efficiency of the communication duration. We call that approach *network-aware grouping*. In the network-aware grouping, basically, clients with low bandwidth tiers are placed in groups of smaller model sizes. Therefore, only the pruned models with smaller sizes face bandwidth bottleneck problems and other pruned models would be executed in shorter communication duration. The overall duration of federated fine-tuning of pruned models would be reduced as well.

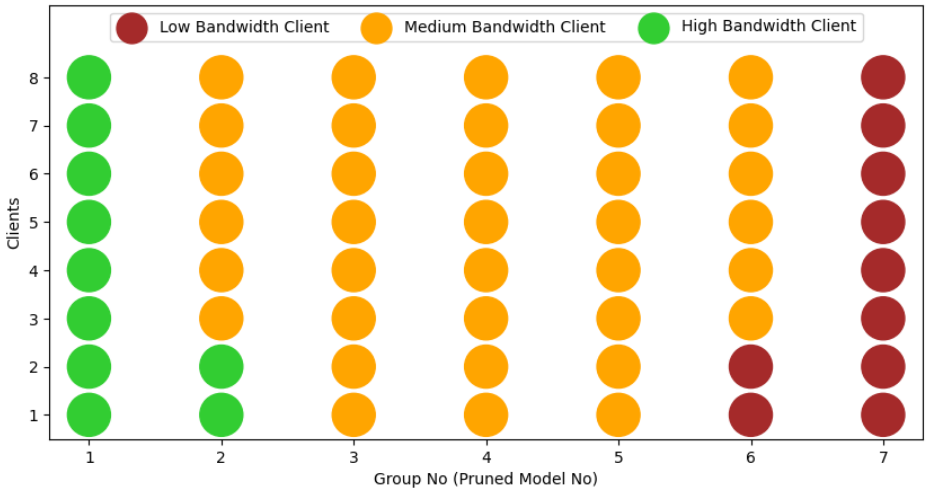


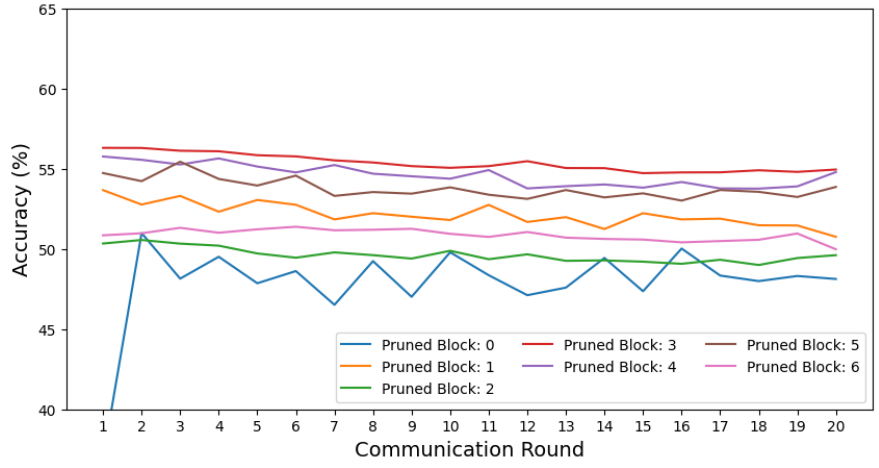
Figure 5.13. Groups generated by Network-aware grouping.

Table 5.3. EMD values of network-aware groups.

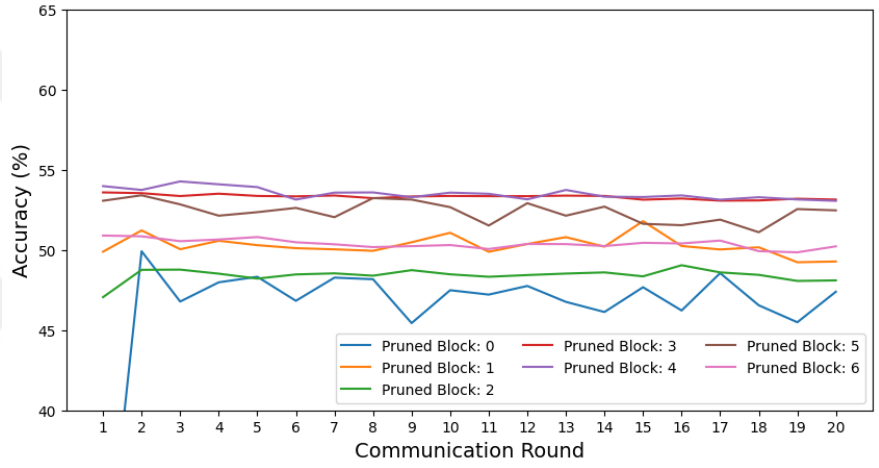
Group No.	**1**	**2**	**3**	**4**	**5**	**6**	**7**
EMD	0.58	0.61	0.65	0.63	0.32	0.61	0.50

Figure 5.13 shows the placements of clients to groups by their network properties. Since medium and high bandwidth clients are placed to the models with greater sizes, low bandwidth clients are stacked to the right, to the models with smaller sizes. As a result the bottleneck only occurred in the groups of 5^{th} and 6^{th} models as shown in Figure 5.13.

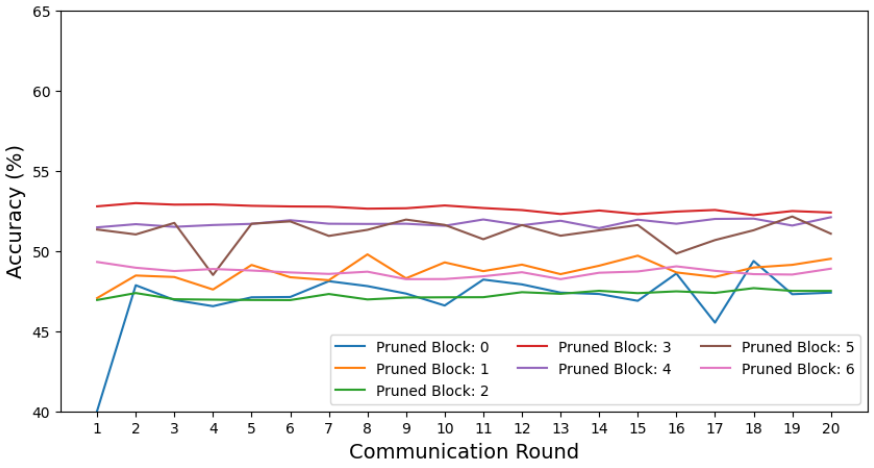
Grouping of clients cannot be conducted without considering the class distributions of clients and built groups. The EMD values of each group are investigated in Table 5.3. The average EMD value increased from 0.52 to 0.56, and most of the EMD values of clients increased from basic grouping to network-aware grouping. Moreover, as shown in Figure 5.14, accuracies for different iterations and pruned models decreased as well.



(a)

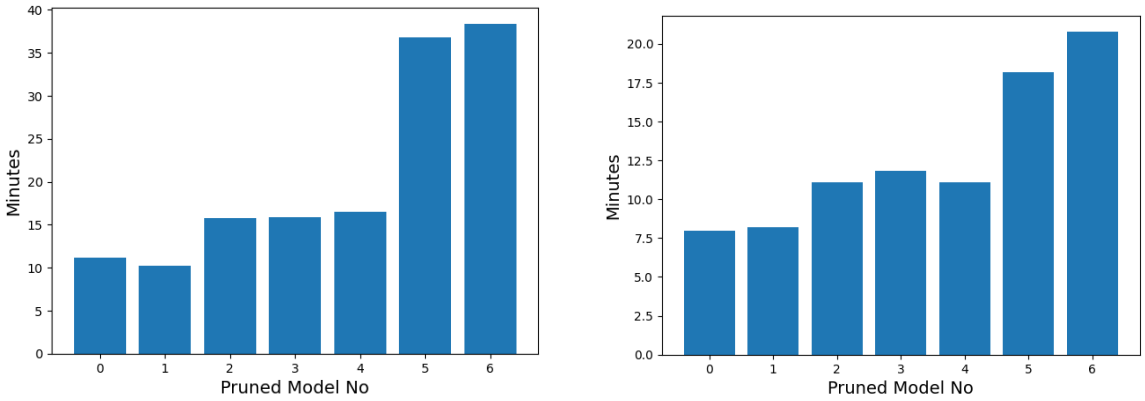


(b)



(c)

Figure 5.14. Federated fine-tuning accuracies of each model generated with network-aware grouping in (a) iteration 1 (b) iteration 6 and (c) iteration 15.



(a) (b)

Figure 5.15. Overall durations of one communication round in (a) 1st and (b) 10th iterations for pruned models with network-optimized grouping.

Round duration for pruned models is reduced when the bottleneck is removed. However, the bottleneck still happens for the models with smaller sizes in an iteration. For a sequential federated fine-tuning of pruned models, the overall duration would be reduced significantly, for a parallel federated fine-tuning of pruned models the duration of a neural network pruning iteration the duration is decided by the pruned model whose federated fine-tuning duration is the longest. One round duration with network-aware grouping is shown in Figure 5.15(a) and Figure 5.15(b).

From basic grouping to network-aware grouping, the accuracy has dropped for both training and validation accuracies. In Figure 5.14(a), Figure 5.14(b) and Figure 5.14(c), accuracy for different iterations in federated fine-tuning process has been shown and has a similar change with basic grouping. Even though there is a slight fall between EMD values calculated in each approach, the reduction in accuracy is more significant than that fall. The accuracy of the best model in each iteration is not affected by only the non-IID level of the group dataset, and also the pruned layers of each iteration differ in different grouping approaches. Since the validation accuracies differ for each pruned model with the change of client groups, the pruned blocks in each iteration change as well. Therefore the best model performance in each iteration is likely to differ.

5.6. Network and Distribution Optimized Client Grouping

In this section, our proposed client grouping algorithm NetDOG has been utilized in the federated neural network pruning. The algorithm parameters *MAX_ITER* and *COEFF* are set as 10 and 0.9 respectively. That means the algorithm can run a maximum of 10 iterations and if the reduction in average EMD value after the transfer is higher than 10% the transfer is permitted. After the EMD values are reduced through the iterations, the transfers between groups become less likely to happen because the reduction possibility decreases. The initial group configuration is set as the simple network-aware grouping in Figure 5.13.

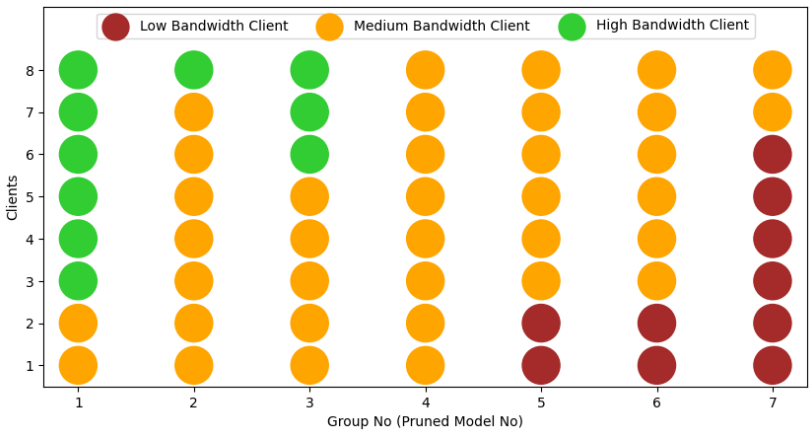
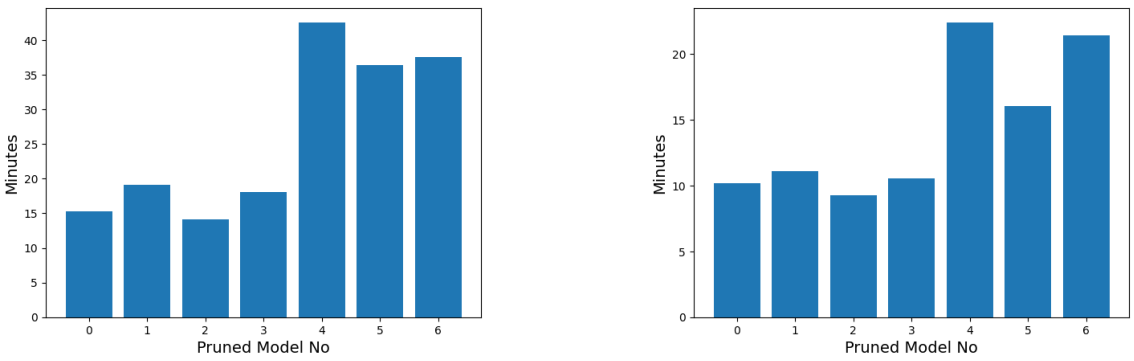
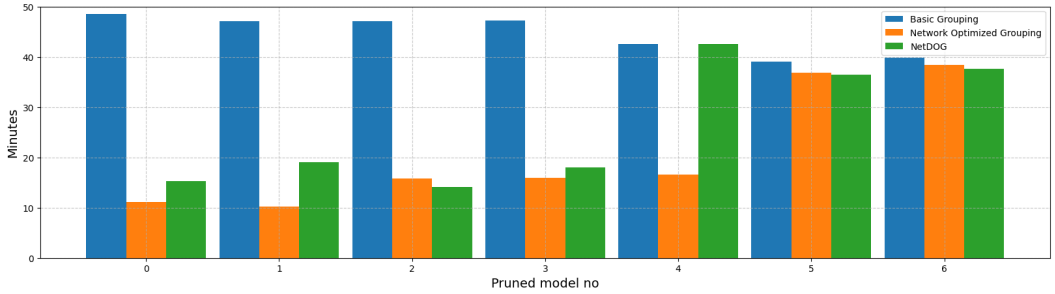


Figure 5.16. Groups generated by NetDOG.

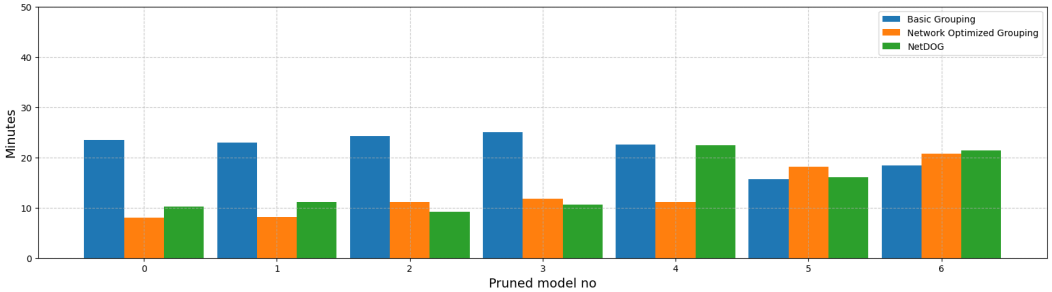


(a) (b)

Figure 5.17. Overall durations of one communication round in (a) 1st and (b) 10th iterations for pruned models with NetDOG grouping.



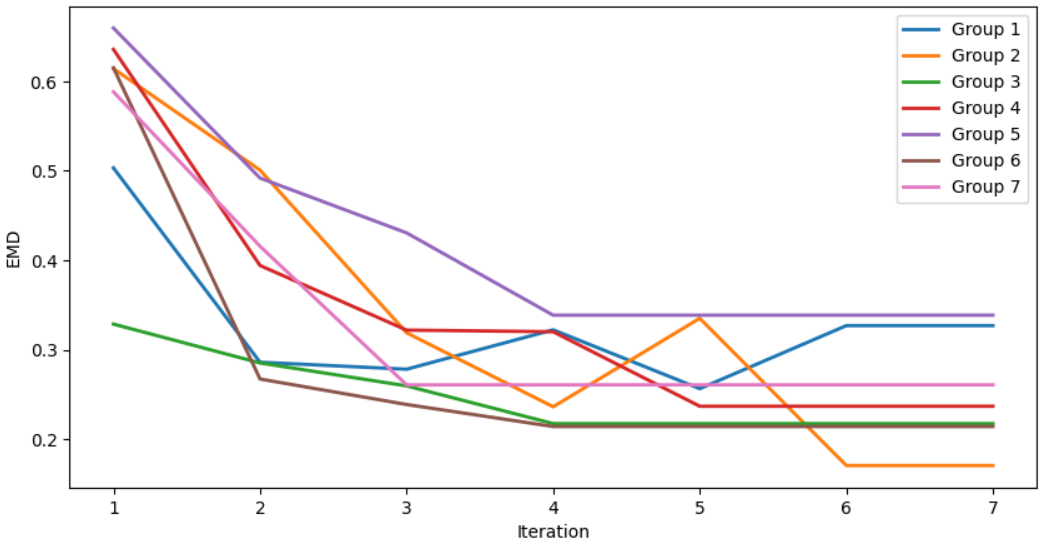
(a)



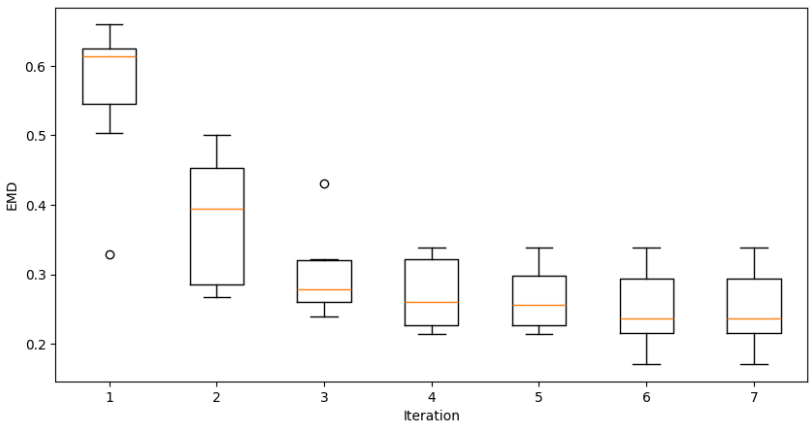
(b)

Figure 5.18. Comparison of overall durations of pruned models across grouping methods for one communication round in (a) 1^{st} and (b) 10^{th} iterations.

The NetDOG has lasted 7 iterations in optimizing network bandwidths and the EMD values in order words non-IIDness of groups. Through the iterations, multiple transfers have occurred and the low bandwidth bottleneck is distributed from 2 groups to 3 groups as shown in Figure 5.16. That result still holds the low bandwidth clients to groups assigned to the three smallest pruned models. Consequently, the one round duration of the pruned model has changed, especially one more group sacrificed by placing a low bandwidth bottleneck compared to network-aware grouping. As shown in Figure 5.17(a) and Figure 5.17(b), pruned models with the numbers 4, 5, and 6 lasted longer since their groups have clients with low bandwidth. Figure 5.18(a) and Figure 5.18(b) compare NetDOG with the other grouping approaches, our approach still requires less time than basic grouping, however, since one more group is sacrificed the duration of an iteration increases compared to network-aware grouping.



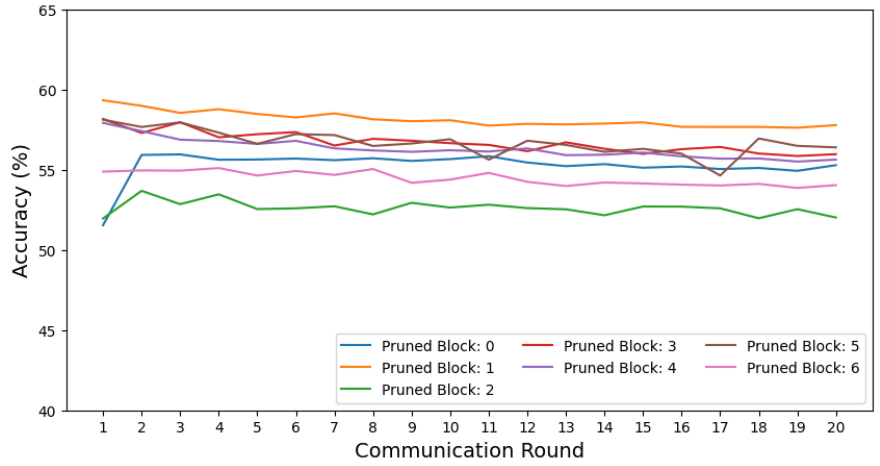
(a)



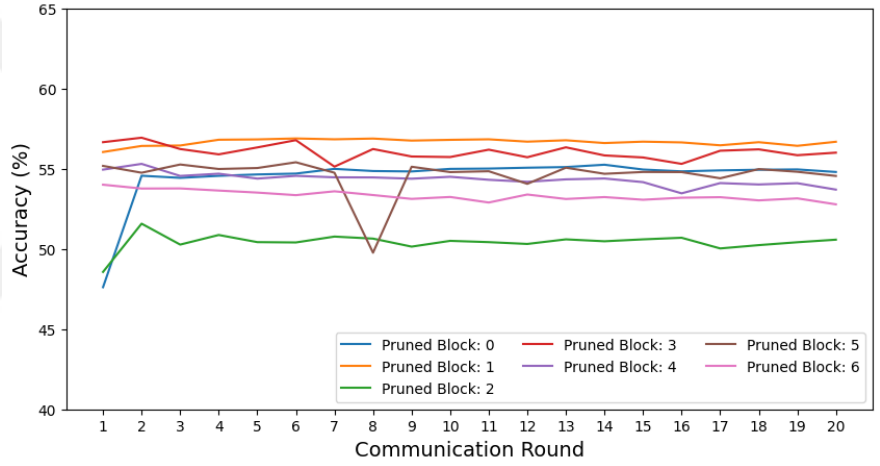
(b)

Figure 5.19. Change in EMD of groups through iterations in terms of (a) EMD by each group and (b) change in average EMD.

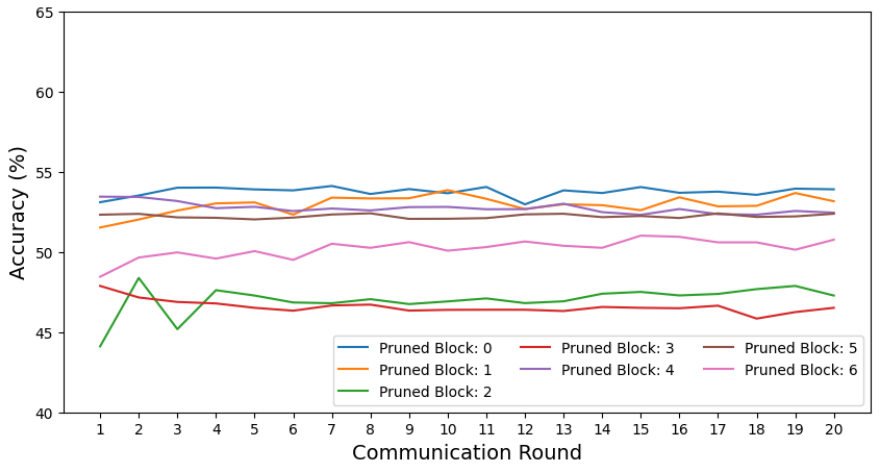
However, with the help of NetDOG EMD values have been balanced and reduced significantly compared to basic and network-aware grouping. As shown in Figure 5.19(a) and Figure 5.19(b), EMD values and the average EMD value are significantly reduced through the iterations while sacrificing one group by placing low bandwidth clients in it.



(a)



(b)



(c)

Figure 5.20. Federated fine-tuning accuracies of each model generated with NetDOG in (a) iteration 1 (b) iteration 6 and (c) iteration 15.

Table 5.4. Comparison of EMD values for grouping approaches.

Grouping Type	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Avg**
Basic	0.58	0.57	0.62	0.49	0.42	0.41	0.67	0.52
Network-Aware	0.58	0.61	0.65	0.63	0.32	0.61	0.50	0.56
NetDOG	0.32	0.17	0.21	0.23	0.33	0.21	0.26	**0.25**

The comparison of EMD values of different grouping strategies is given in Table 5.4. Consequently, our algorithm is effective in reducing the imbalance situation of class distributions in other words level of non-IID. The group sacrifice made by the algorithm can be controlled by the coefficient parameter, setting the coefficient to 1 would permit every operation to be done but setting it to 0 would prohibit transfers of client pairs with different bandwidth tiers. As an example, in Figure 5.16, two low bandwidth clients shifted from other groups. In the initial condition group 5 does not have low bandwidth clients, which means there were 2 transfers between different bandwidth tiers. The coefficient was set to 0.9 in that experiment so those transfers were allowed because they reduced the average EMD value by more than 10 percent. In order to restrict the transfers or maybe prevent the transfers, the coefficient would be reduced.

Table 5.5. Comparison of validation accuracy across pruned models.

Model	**Basic Grouping**	**Network-Aware Grouping**	**NetDOG**
87% AlexNet	55.42	54.19	**56.13**
70% AlexNet	55.14	53.74	**55.48**
60% AlexNet	53.60	52.69	**55.16**
55% AlexNet	52.59	52.38	**53.92**

With the help of the NetDOG, the overall federate neural network pruning is executed and pruned models have been evaluated. Decreased EMD values of groups lead to achieving competitive accuracies. As shown in Table 5.5, NetDOG showed

slightly better performance than basic grouping for some of the pruned models through the iterations. Even though the performances of pruned models are reduced through the iterations, the performance of our algorithm shows better performance. Network-aware grouping never showed better performance through the different pruned models.

Another aspect is that, as illustrated in Figure 5.20(a), Figure 5.20(b) and Figure 5.20(c) , the accuracies of pruned models in a single iteration showed closer results when NetDOG is used than when other strategies were used. That means grouping with more balanced class distributions leads all models to perform better, in another perspective, our algorithm prevent some pruned models to be failed.

6. CONCLUSION

In this study, we presented an extensive exploration of network-aware federated neural architecture search, by offering novel solutions to challenges in federated learning in an iterative neural network pruning algorithm. The development of our federated neural network pruning framework represents significant benefits, addressing the complexities of DNN pruning in a distributed setting. Even though there are works in the field of federated neural architecture search, our framework is also able to emulate the network over clients and servers in federated learning.

Firstly, we have built an open-source federated neural network pruning framework with network emulation in order to investigate the effect of network properties such as bandwidth, delay, and jitter on the performance of federated learning and the iterative neural network pruning algorithm. We have successfully ensembled an iterative neural network pruning algorithm, federated learning framework, and network emulator. In traditional federated learning approaches, models with the same architecture and parameters are shared with clients therefore clients have the information of model architecture beforehand. Our case changes in architecture and parameters, require sharing knowledge of those aspects with clients as well. Therefore, the communication of the pruning algorithm and federated environment needs to be developed. We have implemented an open-source DNN model serialization method that serializes both architecture and weight knowledge which is going to be distributed to clients. Moreover, communication between servers and clients is managed by the network emulator. In the network emulator, we have implemented dynamic bandwidth changes for client network properties and successfully simulated the real-world.

Our key results show that, without the consideration of network conditions bandwidth federated learning systems do not represent the real-world scenarios. The bandwidth of clients has a significant effect on communication duration while global model weight and its architecture are being shared with clients and sending local updates

to the server. That communication overhead increases the overall time required to converge to the accurate global model. Varying network conditions also create a bottleneck in federated learning since low-bandwidth clients require a longer time while better-bandwidth clients are waiting for those clients. Our framework is able to emulate all those experimental aspects while providing a distributed neural network pruning framework.

Furthermore, our contribution provides a solution to a key challenge in federated learning settings. As explicitly investigated in our study, low bandwidth clients create bottlenecks in communication rounds of federated learning. Our proposed solution to that problem extends to the perspective of client selection. Our algorithm NetDOG provides client grouping, where our novel approach considers both the non-IID nature of client data and the network bandwidth bottlenecks. We have utilized the EMD metric for quantifying the level of non-IID of clients and groups and deciding which transfer operation between neighboring groups would be done. By optimizing client organization based on class distributions, we improved the efficiency and balance of federated learning updates in the aspects of both accuracy and round duration. That approach would be beneficial for the feasibility of iterative federated learning applications in real-world environments where clients have various network conditions and non-IID local data.

In essence, this thesis contributes to the advancement of network-aware federated neural architecture search by presenting a holistic framework for model compression, introducing an intelligent client grouping algorithm, and exposing the details between network conditions and federated learning efficiency. In further developments, behaviors of other neural architecture search algorithms on the network emulated federated learning framework, the effect of more complex non-IID datasets on client grouping, and different network condition properties, can be investigated.

REFERENCES

1. Chacko, A. and T. Hayajneh, "Security and Privacy Issues with IoT in Healthcare", *Endorsed Transactions on Pervasive Health and Technology*, Vol. 4, No. 14, pp. 1–7, 2018.

2. LeCun, Y., Y. Bengio and G. Hinton, "Deep Learning", *Nature*, Vol. 521, No. 7553, pp. 436–444, 2015.

3. Liu, S., H. Zhang and Y. Jin, "A Survey on Computationally Efficient Neural Architecture Search", *Journal of Automation and Intelligence*, Vol. 1, No. 1, p. 100002, 2022.

4. Zoph, B. and Q. V. Le, "Neural Architecture Search with Reinforcement Learning", ArXiv:1611.01578, 2016.

5. Elsken, T., J. H. Metzen and F. Hutter, "Neural Architecture Search: A Survey", *The Journal of Machine Learning Research*, Vol. 20, No. 5, pp. 1997–2017, 2019.

6. Chen, M., J. Yang, J. Zhou, Y. Hao, J. Zhang and C.-H. Youn, "5G-Smart Diabetes: Toward Personalized Diabetes Diagnosis with Healthcare Big Data Clouds", *Communications Magazine*, Vol. 56, No. 4, pp. 16–23, 2018.

7. Brewster, C., I. Roussaki, N. Kalatzis, K. Doolin and K. Ellis, "IoT in Agriculture: Designing a Europe-Wide Large-Scale Pilot", *Communications Magazine*, Vol. 55, No. 9, pp. 26–33, 2017.

8. Muthuramalingam, S., A. Bharathi, S. Rakesh Kumar, N. Gayathri, R. Sathiyaraj and B. Balamurugan, "IoT Based Intelligent Transportation System (IoT-ITS) for Global Perspective: A Case Study", *Internet of Things and Big Data Analytics for Smart Generation*, Vol. 154, No. 3, pp. 279–300, 2019.

9. Chakrabarty, S. and D. W. Engels, "A secure IoT architecture for Smart Cities", *Consumer Communications & Networking Conference*, Las Vegas, USA, pp. 812–813, 2016.

10. Cimpanu, C., "Thousands of IOT medical devices found vulnerable to online attacks", 2015, `https://news.softpedia.com/news/thousands-of-iot-medical-devices-found-vulnerable-to-online-attacks-493144.shtml`, accessed on December 12, 2023.

11. Albrecht, J. P., "How the GDPR Will Change the World", *European Data Protection Law Review*, Vol. 2, No. 3, pp. 287–289, 2016.

12. Zhang, C., Y. Xie, H. Bai, B. Yu, W. Li and Y. Gao, "A Survey on Federated Learning", *Knowledge-Based Systems*, Vol. 216, No. 4, p. 106775, 2021.

13. McMahan, H. B., E. Moore, D. Ramage, S. Hampson and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data", ArXiv:1602.05629, 2017.

14. Tang, J., D. Sun, S. Liu and J. Gaudiot, "Enabling Deep Learning on IoT Devices", *Computer*, Vol. 50, No. 10, pp. 92–96, 2017.

15. Feurer, M., K. Eggensperger, S. Falkner, M. T. Lindauer and F. Hutter, "Practical Automated Machine Learning for the AutoML Challenge 2018", *International Workshop on Automatic Machine Learning at International Conference on Machine Learning*, Stockholm, Sweden, pp. 1189–1232, 2018.

16. Truong, A., A. Walters, J. Goodsitt, K. Hines, C. Bruss and R. Farivar, "Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools", *International Conference on Tools with Artificial Intelligence*, Los Alamitos, USA, pp. 1471–1479, 2019.

17. He, X., K. Zhao and X. Chu, "AutoML: A Survey of the State-of-the-Art",

ArXiv:1908.00709, 2021.

18. Tan, M. and Q. Le, "Efficientnet: Rethinking Model Scaling for Convolutional Neural Networks", *International Conference on Machine Learning*, Long Beach, USA, pp. 6105–6114, 2019.

19. Li, G., M. Xu, S. Giancola, A. Thabet and B. Ghanem, "LC-NAS: Latency Constrained Neural Architecture Search for Point Cloud Networks", *International Conference on 3D Vision*, Los Alamitos, USA, pp. 1–11, 2022.

20. Xie, L., X. Chen, K. Bi, L. Wei, Y. Xu, L. Wang, Z. Chen, A. Xiao, J. Chang, X. Zhang and Q. Tian, "Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap", *Computing Surveys*, Vol. 54, No. 9, pp. 1–37, 2021.

21. White, C., M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey and F. Hutter, "Neural Architecture Search: Insights from 1000 Papers", ArXiv:2301.08727, 2023.

22. Jin, H., Q. Song and X. Hu, "Auto-Keras: An Efficient Neural Architecture Search System", *International Conference on Knowledge Discovery & Data Mining*, Anchorage, USA, p. 1946–1956, 2019.

23. Xu, S., A. Huang, L. Chen and B. Zhang, "Convolutional Neural Network Pruning: A Survey", *39th Chinese Control Conference*, Shenyang, China, pp. 7458–7463, 2020.

24. Raja, K. B., R. Raghavendra and C. Busch, "Obtaining Stable Iris Codes Exploiting Low-Rank Tensor Space and Spatial Structure Aware Refinement for Better Iris Recognition", *International Conference on Biometrics*, Crete, Greece, pp. 1–8, 2019.

25. Liu, Z. S. M. C. K.-T., Zechunand Shen, "ReActNet: Towards Precise Binary Neural Network with Generalized Activation Functions", *European Conference on*

Computer Vision, Glasgow, UK, pp. 143–159, 2020.

26. Reed, R., "Pruning Algorithms-A Survey", *Transactions on Neural Networks*, Vol. 4, No. 5, pp. 740–747, 1993.

27. Luo, J.-H., J. Wu and W. Lin, "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression", ArXiv:1707.06342, 2017.

28. Yang, A. C. B. Z.-X. G. A. S. M. S. V. A. H., Tien-Juand Howard, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications", *European Conference on Computer Vision*, Munich, Germany, pp. 289–304, 2018.

29. Satyanarayanan, M., P. Bahl, R. Cáceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", *Pervasive Computing*, Vol. 8, No. 4, pp. 14–23, 2009.

30. Konecný, J., H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency", ArXiv:1610.05492, 2016.

31. Ji, S., T. Saravirta, S. Pan, G. Long and A. Walid, "Emerging Trends in Federated Learning: From Model Fusion to Federated X Learning", ArXiv:2102.12920, 2021.

32. Sahu, A. K., T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar and V. Smith, "Federated Optimization in Heterogeneous Networks", ArXiv:1812.06127, 2018.

33. Zhao, Y., M. Li, L. Lai, N. Suda, D. Civin and V. Chandra, "Federated Learning with Non-IID Data", ArXiv:1806.00582, 2018.

34. Sturluson, S. P., S. Trew, L. M. noz González, M. Grama, J. Passerat-Palmbach, D. Rueckert and A. Alansary, "FedRAD: Federated Robust Adaptive Distillation", ArXiv:2112.01405, 2021.

35. Chen, L.-C., M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam and J. Shlens, "Searching for Efficient Multi-Scale Architectures for Dense Image Prediction", ArXiv:1809.04184, 2018.

36. Du, X., T.-Y. Lin, P. Jin, G. Ghiasi, M. Tan, Y. Cui, Q. V. Le and X. Song, "SpineNet: Learning Scale-Permuted Backbone for Recognition and Localization", *Conference on Computer Vision and Pattern Recognition*, Seattle, USA, pp. 11589–11598, 2020.

37. So, D. R., C. Liang and Q. V. Le, "The Evolved Transformer", ArXiv:1901.11117, 2019.

38. Zoph, B., V. Vasudevan, J. Shlens and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition", ArXiv:1707.07012, 2017.

39. Pham, H., M. Guan, B. Zoph, Q. Le and J. Dean, "Efficient Neural Architecture Search via Parameters Sharing", *International Conference on Machine Learning*, Stockholm, Sweden, pp. 4095–4104, 2018.

40. Angeline, P., G. Saunders and J. Pollack, "An Evolutionary Algorithm That Constructs Recurrent Neural Networks", *Transactions on Neural Networks*, Vol. 5, No. 2, pp. 54–65, 1994.

41. Stanley, K. O. and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies", *Evolutionary Computation*, Vol. 10, No. 2, pp. 99–127, 2002.

42. Real, E., A. Aggarwal, Y. Huang and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search", *The AAAI Conference on Artificial Intelligence*, Hanolulu, USA, pp. 4780–4789, 2019.

43. Elsken, T., J.-H. Metzen and F. Hutter, "Simple And Efficient Architecture Search for Convolutional Neural Networks", ArXiv:1711.04528, 2017.

44. Elsken, T., J. H. Metzen and F. Hutter, "Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution", ArXiv:1804.09081, 2018.

45. Bender, G., P.-J. Kindermans, B. Zoph, V. Vasudevan and Q. Le, "Understanding and Simplifying One-Shot Architecture Search", *International Conference on Machine Learning*, Stockholm, Sweden, pp. 550–559, 2018.

46. Saxena, S. and J. Verbeek, "Convolutional Neural Fabrics", ArXiv:1606.02492, 2016.

47. Cai, H., C. Gan and S. Han, "Once for All: Train One Network and Specialize it for Efficient Deployment", ArXiv:1908.09791, 2019.

48. Liu, H., K. Simonyan and Y. Yang, "DARTS: Differentiable Architecture Search", ArXiv:1806.09055, 2018.

49. Vadera, S. and S. Ameen, "Methods for Pruning Deep Neural Networks", *Access*, Vol. 10, No. 2, pp. 63280–63300, 2022.

50. Zhou, Z., W. Zhou, R. Hong and H. Li, "Online Filter Weakening and Pruning for Efficient Convnets", *International Conference on Multimedia and Expo*, San Diego, USA, pp. 1–6, 2018.

51. Lin, S., R. Ji, Y. Li, Y. Wu, F. Huang and B. Zhang, "Accelerating Convolutional Networks via Global Dynamic Filter Pruning", *International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, pp. 2425–2432, 2018.

52. Zhang, L., Z. Tan, J. Song, J. Chen, C. Bao and K. Ma, "SCAN: A Scalable Neural Networks Framework Towards Compact and Efficient Models", ArXiv:1906.03951, 2019.

53. Gong, C., Y. Chen, Y. Lu, T. Li, C. Hao and D. Chen, "VecQ: Minimal Loss DNN Model Compression With Vectorized Weight Quantization", *Transactions*

on Computers, Vol. 70, No. 5, pp. 696—710, 2021.

54. Lin, M., R. Ji, Y. xin Zhang, B. Zhang, Y. Wu and Y. Tian, "Channel Pruning via Automatic Structure Search", ArXiv:2001.08565, 2020.

55. Li, H., X. Yue, Z. Wang, Z. Chai, W. Wang, H. Tomiyama and L. Meng, "Optimizing the Deep Neural Networks by Layer-Wise Refined Pruning and the Acceleration on FPGA", *Computational Intelligence and Neuroscience*, Vol. 2022, No. 1, pp. 1687–1709, 2022.

56. Chung, G. and C. Won, "Filter pruning by image channel reduction in pre-trained convolutional neural networks", *Multimedia Tools and Applications*, Vol. 80, No. 3, p. 30817–30826, 2021.

57. Li, Z., H. Li and L. Meng, "Model Compression for Deep Neural Networks: A Survey", *Computers*, Vol. 12, No. 3, p. 60, 2023.

58. Han, S., J. Pool, J. Tran and W. J. Dally, "Learning both weights and connections for efficient neural networks", *International Conference on Neural Information Processing Systems*, Montreal, Canada, p. 1135–1143, 2015.

59. Yang, T.-J., Y.-H. Chen and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning", *Conference on Computer Vision and Pattern Recognition*, Honolulu, USA, pp. 6071–6079, 2017.

60. Banner, R., Y. Nahshan, E. Hoffer and D. Soudry, "ACIQ: Analytical Clipping for Integer Quantization of Neural Networks", ArXiv:1810.05723, 2018.

61. Jacob, B., S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference", *Conference on Computer Vision and Pattern Recognition*, Honolulu, USA, pp. 2704–2713, 2017.

62. Tailor, S. A., J. Fernandez-Marques and N. D. Lane, "Degree-Quant: Quantization-Aware Training for Graph Neural Networks", ArXiv:2008.05000, 2020.

63. Hinton, G. E., O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network", ArXiv:1503.02531, 2015.

64. Tian, Y., D. Krishnan and P. Isola, "Contrastive Representation Distillation", ArXiv:1910.10699, 2019.

65. Hard, A., K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon and D. Ramage, "Federated Learning for Mobile Keyboard Prediction", ArXiv:1811.03604, 2018.

66. Yang, W., Y. Zhang, K. Ye, L. Li and C.-Z. Xu, "FFD: A Federated Learning Based Method for Credit Card Fraud Detection", *Big Data – BigData 2019*, San Diego, USA, pp. 18–32, 2019.

67. Rieke, N., J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein *et al.*, "The Future of Digital Health with Federated Learning", *Digital Medicine*, Vol. 3, No. 1, pp. 119–133, 2020.

68. Briggs, C., Z. Fan and P. Andras, "A Review of Privacy Preserving Federated Learning for Private IoT Analytics", ArXiv:2004.11794, 2020.

69. Hsieh, K., A. Phanishayee, O. Mutlu and P. Gibbons, "The Non-IID Data Quagmire of Decentralized Machine Learning", *International Conference on Machine Learning*, Vienna, Austria, pp. 4387–4398, 2020.

70. Itahara, S., T. Nishio, Y. Koda, M. Morikura and K. Yamamoto, "Distillation-Based Semi-Supervised Federated Learning for Communication-Efficient Collaborative Training With Non-IID Private Data", *Transactions on Mobile Computing*, Vol. 22, No. 1, pp. 191–205, 2020.

71. Jeong, E., S. Oh, H. Kim, J. Park, M. Bennis and S.-L. Kim, "Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data", ArXiv:1811.11479, 2018.

72. Cai, L., D. Lin, J. Zhang and S. Yu, "Dynamic Sample Selection for Federated Learning with Heterogeneous Data in Fog Computing", *International Conference on Communications*, Dublin, Ireland, pp. 1–6, 2020.

73. Wang, Z., Y. Zhu, D. Wang and Z. Han, "FedACS: Federated Skewness Analytics in Heterogeneous Decentralized Data Environments", *International Symposium on Quality of Service*, Tokyo, Japan, pp. 1–10, 2021.

74. Zeng, Y., X. Wang, J. Yuan, J. Zhang and J. Wan, "Local Epochs Inefficiency Caused by Device Heterogeneity in Federated Learning", *Wireless Communications and Mobile Computing*, Vol. 2022, No. 1, pp. 1–15, 2022.

75. Ma, Q., Y. Xu, H. Xu, Z. Jiang, L. Huang and H. Huang, "FedSA: A Semi-Asynchronous Federated Learning Mechanism in Heterogeneous Edge Computing", *Journal on Selected Areas in Communications*, Vol. 39, No. 12, pp. 3654–3672, 2021.

76. Reddi, S. J., Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Koneý, S. Kumar and H. B. McMahan, "Adaptive Federated Optimization", ArXiv:2003.00295, 2020.

77. Ma, X., J. Zhu, Z. Lin, S. Chen and Y. Qin, "A State-of-The-Art Survey on Solving Non-IID data in Federated Learning", *Future Generation Computer Systems*, Vol. 135, No. C, p. 244–258, 2022.

78. Teng, Y., Z. Li, L. Yin and N. Wu, "State-Based Differential Privacy Verification and Enforcement for Probabilistic Automata", *Mathematics*, Vol. 11, No. 8, p. 1853, 2023.

79. McMahan, H. B., D. Ramage, K. Talwar and L. Zhang, "Learning Differentially

Private Language Models Without Losing Accuracy", ArXiv:1710.06963, 2017.

80. Ramaswamy, S., O. Thakkar, R. Mathews, G. Andrew, H. B. McMahan and F. Beaufays, "Training Production Language Models without Memorizing User Data", ArXiv:2009.10031, 2020.

81. Sun, L. and L. Lyu, "Federated Model Distillation with Noise-Free Differential Privacy", ArXiv:2009.05537, 2020.

82. Li, D. and J. Wang, "FedMD: Heterogenous Federated Learning via Model Distillation", ArXiv:1910.03581, 2019.

83. Liu, L., J. Zhang, S. Song and K. B. Letaief, "Communication-Efficient Federated Distillation with Active Data Sampling", *International Conference on Communications*, Seoul, South Korea, pp. 201–206, 2022.

84. Jiang, D., C. Shan and Z. Zhang, "Federated Learning Algorithm Based on Knowledge Distillation", *International Conference on Artificial Intelligence and Computer Engineering*, Beijing, China, pp. 163–167, 2020.

85. Wu, C., F. Wu, L. Lyu, Y. Huang and X. Xie, "Communication Efficient Federated Learning via Knowledge Distillation", *Nature Communications*, Vol. 13, No. 4, p. 2032, 2022.

86. Zhu, H., H. Zhang and Y. Jin, "From Federated Learning to Federated Neural Architecture Search: A Survey", *Complex & Intelligent Systems*, Vol. 7, No. 2, pp. 639–657, 2021.

87. Zhu, H. and Y. Jin, "Multi-Objective Evolutionary Federated Learning", *Transactions on Neural Networks and Learning Systems*, Vol. 31, No. 4, pp. 1310–1322, 2020.

88. Zhu, H. and Y. Jin, "Real-Time Federated Evolutionary Neural Architecture

Search", *Transactions on Evolutionary Computation*, Vol. 26, No. 7, pp. 364–378, 2021.

89. He, C., M. Annavaram and S. Avestimehr, "FedNAS: Federated Deep Learning via Neural Architecture Search", ArXiv:2004.08546, 2020.

90. Xu, M., Y. Zhao, K. Bian, G. Huang, Q. Mei and X. Liu, "Federated Neural Architecture Search", ArXiv:2002.06352, 2020.

91. Trick, M. A., "A Linear Relaxation Heuristic for The Generalized Assignment Problem", *Naval Research Logistics*, Vol. 39, No. 4, pp. 137–151, 1992.

92. Dudziak, L., S. Laskaridis and J. Fernandez-Marques, "FedorAS: Federated Architecture Search under System Heterogeneity", ArXiv:2206.11239, 2022.

93. Krizhevsky, A., I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Communications*, Vol. 60, No. 6, p. 84–90, 2017.

94. Beutel, D. J., T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet and P. P. B. de Gusmão, "Flower: A Friendly Federated Learning Research Framework", ArXiv:2007.14390. 2020.

95. Ahrenholz, J., C. Danilov, T. R. Henderson and J. H. Kim, "CORE: A Real-Time Network Emulator", *Military Communications Conference*, San Diego, USA, pp. 1–7, 2008.