FedNAS: Federated Deep Learning via Neural Architecture Search

Chaoyang He Murali Annavaram Salman Avestimehr
University of Southern California
chaoyang.he@usc.edu annavara@usc.edu avestime@usc.edu

Abstract

Federated Learning (FL) has been proved to be an effective learning framework when data cannot be centralized due to privacy, communication costs, and regulatory restrictions. When training deep learning models under an FL setting, people employ the predefined model architecture discovered in the centralized environment. However, this predefined architecture may not be the optimal choice because it may not fit data with non-identical and independent distribution (non-IID). Thus, we advocate automating federated learning (AutoFL) to improve model accuracy and reduce the manual design effort. We specifically study AutoFL via Neural Architecture Search (NAS), which can automate the design process. We propose a Federated NAS (FedNAS) algorithm to help scattered workers collaboratively searching for a better architecture with higher accuracy. We also build a system based on FedNAS. Our experiments on non-IID dataset show that the architecture searched by FedNAS can outperform the manually predefined architecture [1]

1. Introduction

While CNN (Convolutional Neural Network) is capable of performing various computer vision tasks, it consumes a significant amount of data to achieve state-of-the-art performance. In the case of COVID-19, results from CT imaging, an effective method used to diagnose coronavirus-induced pneumonia, varies with the age and immunity status of the patient, as well as the disease stage at the time of scanning, underlying health conditions, and other drug interventions used at the time. Thus, significant amounts of CT image data from patients across hospitals around the world are essential for rapid and accurate diagnosis and treatment. However, centralizing such small and scattered heterogeneous data from various organizations is challenging due to privacy and confidentiality concerns, high communication and storage costs, protection of intellectual property, regulatory restrictions, and legal constraints. Consequently, a promising approach to solve this problem is Federated Learning (FL) [10], a decentralized learning framework which can train models without sharing the raw data with the collaborating nodes. For example, in Figure 1, each hospital can each learn a CNN model without disclosing its own raw CT images.

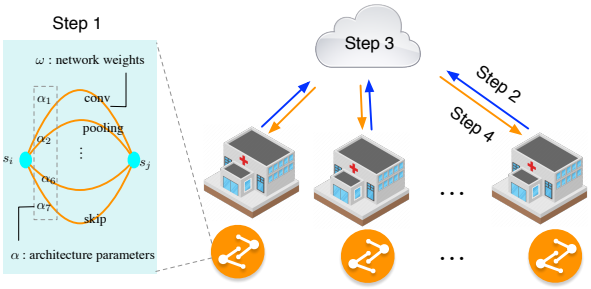


Figure 1: Federated Neural Architecture Search (*step 1*: search locally; *step 2*: sending the gradient of α and ω to the server side; *step 3*: merge gradient to get global α and ω; *step 4*: synchronize updated α and ω to each client.)

However, as [8] points out, when training deep neural networks under the FL setting where the data is non-IID (non-identical and independent distribution), using the predefined model architecture may not be the optimal design choice. Since the data distribution is invisible to researchers, to find a better model architecture with higher accuracy, developers must design or choose multiple architectures, then tune hyperparameters remotely to fit the scattered data. This process is extremely expensive because attempting many rounds of training on edge devices results in a remarkably higher communication cost and on-device computational burden than the data center environment.

Due to these challenges, we advocate automating federated learning, which we term as AutoFL (Automated Federated Learning). In this work, we specifically study AutoFL via Neural Architecture Search (NAS), which automates the design process of the model architecture without inefficient manual attempts. In the centralized data environment, NAS

[1] This paper was accepted to CVPR 2020 workshop on neural architecture search and beyond for representation learning. We released the source code at https://fedml.ai.

significantly outperforms many manually designed architectures [7]. However, it is still not clear whether NAS can really help boost the model performance in a heterogeneous distributed data environment like FL. Therefore, we aim to answer a fundamental question: *can we design a collaborative NAS framework that helps improve model performance in the federated deep learning setting, in which the data distribution is non-identical and independent?*

To answer this question, we consider the cross-organization scenario, in which each client in the network is a GPU-equipped edge server located in an organization (e.g., the hospital shown in Figure 1). We then propose Federated NAS (FedNAS) to search architectures among edge servers collaboratively. As the process shown in Figure 1, in FedNAS, we first utilize the gradient-based method MiLeNAS [4] as local searcher on the local data of each worker. MiLeNAS can be easily distributed, and it is efficient in terms of search time. Formally, it formulates NAS as a mixed-level problem: $w = w - \eta_w \nabla_w \mathcal{L}_{\text{tr}}(w, \alpha), \alpha = \alpha - \eta_\alpha \left(\nabla_\alpha \mathcal{L}_{\text{tr}}(w, \alpha) + \lambda \nabla_\alpha \mathcal{L}_{\text{val}}(w, \alpha)\right)$, where w represents the network weight and α determines the neural architecture. $\mathcal{L}_{tr}(w, \alpha)$ and $\mathcal{L}_{val}(w, \alpha)$ denote the loss with respect to training data and validation data with w and α, respectively. After the local search, each worker then synchronizes weights w and architecture α with other workers by weighted aggregation.

We design an AutoFL system based on the FedNAS to evaluate our idea. We construct a non-IID image dataset based on CIFAR10 and deploy our system in a distributed computing environment, which consists of 16 clients and 1 server. Our experiments show that Fed-NAS can search for a better architecture with a higher accuracy in only a few hours compared to *FedAvg* [10], which purely utilizes the state-of-the-art manually designed architecture.

We also summarize our research directions that have the potential to further improve the efficiency and effectiveness of our proposed FedNAS.

2. Related Works

Recently, Neural Architecture Search (NAS) [7] has attracted widespread attention due to its advantages over manually designed models. There are three major NAS methods: evolutionary algorithms, reinforcement learning-based methods, and gradient-based methods [4]. While in the Federated Learning (FL) domain [10, 3], using designed model architectures and optimizing by FedAvg [10] is the main method to improve model performance. To our knowledge, NAS is rarely studied in FL setting. Although [8] first proposed the concept of automating FL via NAS, the concrete method and details are never given. In our work, we propose a FedNAS algorithm and demonstrate its efficacy. Our AutoFL system design is relevant to [1]. However, its system only supports manually designed architectures.

3. Proposed Method

3.1. Problem Definition

In federated learning setting, there are K nodes in the network. Each node has a dataset $\mathcal{D}_k := \left\{\left(x_i^k, y_i\right)\right\}_{i=1}^{N_k}$ which is non IID. When collaboratively training a deep neural networks (DNN) model with K nodes, the objective function is defined as:

$$\min_w f(w, \underbrace{\alpha}_{fixed}) \stackrel{\text{def}}{=} \min_w \sum_{k=1}^{K} \frac{N_k}{N} \cdot \frac{1}{N_k} \sum_{i \in \mathcal{D}_k} \ell(x_i, y_i; w, \underbrace{\alpha}_{fixed}) \tag{1}$$

where w represents the network weight, α determines the neural architecture, and ℓ is the loss function of the DNN model. To minimize the objective function above, previous works choose a fixed model architecture α and then design variant optimization techniques to train the model w.

We propose to optimize the federated learning problem from a completely different angle, optimizing w and α simultaneously. Formally, we can reformulate the objective function as:

$$\min_{w,\alpha} f(w, \alpha) \stackrel{\text{def}}{=} \min_{w,\alpha} \sum_{k=1}^{K} \frac{N_k}{N} \cdot \frac{1}{N_k} \sum_{i \in \mathcal{D}_k} \ell(x_i, y_i; w, \alpha) \tag{2}$$

In other words, for the non-IID dataset scattered across many workers, our goal is to search for an optimal architecture α and related model parameters w to fit the dataset more effectively thus achieve better model performance. In this work, we consider searching for CNN architecture to improve the performance of the image classification task.

3.2. Search Space

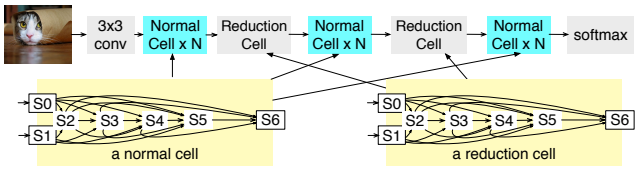


Figure 2: Search Space

Normally, NAS includes three consecutive components: the search space definition, the search algorithm, and the performance estimation strategy [7]. Our search space follows the mixed-operation search space defined in DARTS [9] and MiLeNAS [4], where we search in two shared convolutional cells and then build it up as an entire model architecture (as shown in Figure 2). Inside the cell, to relax the categorical candidate operations between two nodes (e.g., convolution, max pooling, skip connection, zero) to a continuous search space, mixed operation using softmax over

all possible operations is proposed:

$$\bar{o}^{(i,j)}(x) = \sum_{k=1}^{d} \underbrace{\frac{\exp(\alpha_k^{(i,j)})}{\sum_{k'=1}^{d} \exp(\alpha_{k'}^{(i,j)})}}_{p_k} o_k(x) \tag{3}$$

where the weight p_k of the mixed operation $\bar{o}^{(i,j)}(x)$ for a pair of nodes (i, j) is parameterized by a vector $\alpha^{i,j}$. Thus, all architecture operation options inside a network (model) can be parameterized as α. More details are introduced in Appendix B.1.

3.3. Local Search

Following the above-mentioned search space, each worker searches locally by utilizing the mixed-level optimization technique MiLeNAS [4]:

$$\begin{aligned} w &= w - \eta_w \nabla_w \mathcal{L}_{\text{tr}}(w, \alpha) \\ \alpha &= \alpha - \eta_\alpha \left(\nabla_\alpha \mathcal{L}_{\text{tr}}(w, \alpha) + \lambda \nabla_\alpha \mathcal{L}_{\text{val}}(w, \alpha) \right) \end{aligned} \tag{4}$$

where $\mathcal{L}_{tr}(w, \alpha)$ and $\mathcal{L}_{val}(w, \alpha)$ denote the loss with respect to the local training data and validation data with w and α, respectively.

3.4. Federated Neural Architecture Search

We propose FedNAS, a distributed neural architecture search algorithm that aims at optimizing the objective function in Equation 2 under the FL setting. We introduce FedNAS corresponding to four steps in Figure 1: 1) The local searching process: each worker optimizes α and w simultaneously using Eq. 4 for several epochs; 2) All clients send their α and w to the server; 3) The central server aggregates these gradients as follows:

$$\begin{aligned} w_{t+1} &\leftarrow \sum_{k=1}^{K} \frac{N_k}{N} w_{t+1}^k \\ \alpha_{t+1} &\leftarrow \sum_{k=1}^{K} \frac{N_k}{N} \alpha_{t+1}^k \end{aligned} \tag{5}$$

4) The server sends back the updated α and w to clients, and each client updates its local α and w accordingly, before running the next round of searching. This process is summarized in Algorithm 1. After searching, an additional evaluation stage is conducted by using a traditional federated optimization method such as *FedAvg* [10]. Merging search and evaluation into one stage is one of future works (check Appendix C for more details).

3.5. AutoFL System Design

We design an AutoFL system using FedNAS based on `FedML` [2], an open-source research library for federated learning. Details can be found in Appendix A.

Algorithm 1 FedNAS Algorithm.

Initialization: initialize w_0 and α_0; K clients are selected and indexed by k; E is the number of local epochs; T is the number of rounds.
Server executes:
  for each round $t = 0, 1, 2, ..., T-1$ **do**
    for each client k **in parallel do**
      $w_{t+1}^k, \alpha_{t+1}^k \leftarrow$ ClientLocalSearch(k, w_t, α_t)
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{N_k}{N} w_{t+1}^k$
    $\alpha_{t+1} \leftarrow \sum_{k=1}^{K} \frac{N_k}{N} \alpha_{t+1}^k$

ClientLocalSearch(k, w, α): *// Run on client k*
  for e in epoch **do**
    for minibatch in training and validation data **do**
      Update $w = w - \eta_w \nabla_w \mathcal{L}_{\text{tr}}(w, \alpha)$
      Update $\alpha = \alpha - \eta_\alpha \mathcal{L}_{tr}(w, \alpha) + \lambda \mathcal{L}_{val}(w, \alpha)$
  return w, α to server

4. Experiments and Results

4.1. Experimental Setup

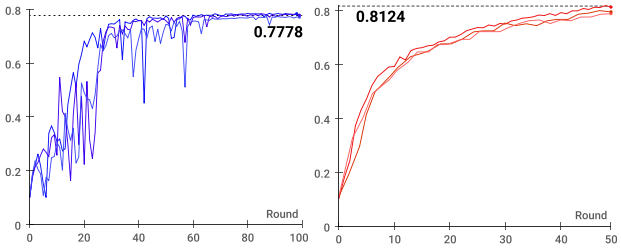
Implementation and Deployment. We set up our experiment in a distributed computing network equipped with GPUs. There are 17 nodes in total, one representing the server-side, and the other 16 nodes representing clients, which can be organizations in the real world (e.g., the hospitals). Each node is a physical server that has an NVIDIA RTX 2080Ti GPU card inside. We deployed the FedNAS system described in Appendix A on each node. Our code implementation is based on PyTorch 1.4.0, MPI4Py [2] 3.0.3 , and Python 3.7.4. For simplicity, we assume all the clients join the training process for every communication round.

Task and Dataset. Our training task is image classification on the CIFAR10 dataset, which consists of 60000 32x32 color images in 10 classes, with 6000 images per class. We generate non-IID (non identical and independent distribution) dataset by splitting the 50000 training images into K clients in an unbalanced manner: sampling $\mathbf{p}_c \sim \text{Dir}_J(0.5)$ and allocating a $\mathbf{p}_{c,k}$ proportion of the training samples of class c to local client k. The 10000 test images are used for a global test after the aggregation of each round. For different methods, we record the global test accuracy as the metric to compare model performance. Since the model performance is sensitive to the data distribution, we fix the non-IID dataset in all experiments for a fair comparison. The actual data distribution used for experiments can be found in Appendix B.2.

Model and Baseline. We compare FedNAS with FedAvg. FedAvg runs on a manually designed architecture, DenseNet[6], which extends ResNet [5], but higher performance and fewer model parameters than ResNet.

[2] https://pypi.org/project/mpi4py/

4.2. Results on non-IID dataset



(a) FedAvg on DenseNet (b) The search stage of FedNAS

Figure 3: Test Accuracy on Non-IID Dataset (multiple runs)

Figure 3a shows the results in a specific non-IID data distribution. For a fair comparison, results are obtained by fine-tuning hyperparameters of each method, and each method is run three times. Figure 3b shows the test accuracy during the searching process. Interestingly, FedNAS can achieve higher accuracy than FedAvg during the searching process (81.24% in Figure 3b; 77.78% in Figure 3a). This further confirms the efficacy of FedNAS. We also evaluate the searched architecture under this data distribution. We found that each run of FedNAS can obtain a higher test accuracy than each run of FedAvg. On average, the architecture searched by FedNAS obtains a test accuracy 4% higher than FedAvg.

Hyperparameters and visualization of the searched architecture can be found in Appendix B.3 and B.4, respectively.

Remark. *We also run experiments on other distributions of non-IID datasets, in which FedNAS is also demonstrated to beat FedAvg, confirming that FedNAS searches for better architectures with a higher model performance.*

4.3. Evaluation of the Efficiency

Method	Search Time	Parameter Size	Hyperparameter
FedAvg (single)	$>$ 3 days	-	rounds = 100 local epochs=20 batch size=64
FedAvg (distributed)	12 hours	20.01M	
FedNAS (single)	33 hours	-	rounds = 50 local epochs=5 batch size=64
FedNAS (distributed)	$<$ **5 hours**	**1.93M**	

Table 1: Efficiency Comparison (16 RTX2080Ti GPUs as clients, and 1 RTX2080Ti as server)

In order to more comprehensively reflect our distributed search overhead, we developed the single-process and distributed version of FedNAS and FedAvg. The single-process version simulates the algorithm by performing a client-by-client search on a single GPU card. As shown in Table 1, compared to FedAvg and manually designed DenseNet, FedNAS can find better architecture with fewer parameters in less time. FedAvg spends more time because it requires more local epochs to converge.

5. Future Works

We introduce our future works in Appendix C.

6. Conclusion

We study automating federated learning (AutoFL) via Neural Architecture Search (NAS) by proposing a Federated NAS (FedNAS) algorithm that can help scatter workers collaboratively searching for a better architecture with higher accuracy. We build an AutoFL system based on FedNAS. Our experiments on the non-IID dataset show that the architecture searched by FedNAS can outperform FedAvg training on the manually designed architecture.

References

[1] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[2] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

[3] Chaoyang He, Conghui Tan, Hanlin Tang, Shuang Qiu, and Ji Liu. Central server free federated learning over single-sided trust social networks. *arXiv preprint arXiv:1910.04956*, 2019.

[4] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[7] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[8] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith

Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[9] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[10] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

A. An AutoFL System based on FedNAS

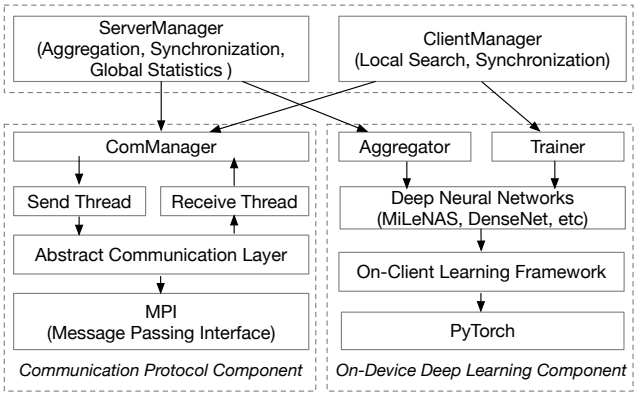


Figure 4: Abstract System Architecture of AutoFL

Our system is built on `FedML` [2], an open-source research library for federated learning. The system architecture is shown in Figure 4. This design separates the communication and the model training into two core components shared by the server and clients. The first is the communication protocol component, which is responsible for low-level communication among the server and clients. The second is the on-device deep learning component, which is built based on the popular deep learning framework PyTorch. These two components are encapsulated as *ComManager*, *Trainer*, and *Aggregator*, providing high-level APIs for the above layers. With the help of these APIs, in *ClientManager*, the client can train or search for better architectures and then send its results to the server-side, while in *ServerManager*, the server can aggregate and synchronize the model architecture and the model parameters with the client-side.

B. More Experiment Details

B.1. Details of the Search Space Definition

We adopt the following 7 operations in our CIFAR-10 experiments: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zero.

The network is formed by stacking convolutional cells multiple times. Cell k takes the outputs of cell $k-2$ and cell $k-1$ as its input. Each cell contains seven nodes: two input nodes, one output node, and the other four intermediate nodes inside the cell. The input of the first intermediate node is set equal to two input nodes, and the other intermediate nodes take all previous intermediate nodes' output as input. The output node concatenates all intermediate nodes' output depth-wise. There are two types of cells: the normal cell and the reduction cell. The reduction cell is designed to reduce the spatial resolution of feature maps, located at 1/3 and 2/3 of the total depth of the network. Architecture parameters determine the discrete operation value between two nodes. All normal cells and all reduction cells share the same architecture parameters $\boldsymbol{\alpha}_{\mathrm{n}}$ and $\boldsymbol{\alpha}_{\mathrm{r}}$, respectively. By this definition, our method alternatively optimizes architecture parameters $(\boldsymbol{\alpha}_{\mathrm{n}}, \boldsymbol{\alpha}_{\mathrm{r}})$ and model weight parameters $\boldsymbol{w}$.

B.2. Details of the heterogeneous distribution on each client (non-IID)

Table 2 shows the actual data distribution used in our experiments. We can see that the sample number of each class in each worker is highly unbalanced. Some classes in a worker even have no samples, and some classes take up most of the proportion (highlighted in the table).

B.3. Hyperparameter Setting

We report important well-tuned hyperparameters used in our experiments. FedNAS searches 50 communication rounds using 5 local searching epochs, with a batch size of 64. For FedAvg, DenseNet201 is used for training, with 100 communication rounds, 20 local epochs, a learning rate of 0.08, and a batch size of 64. Both methods use the same data augmentation techniques that are used in image classification, such as random crops, flips, and normalization. More details and other parameter settings can be found in our source code.

B.4. Visualization of the Search Architecture

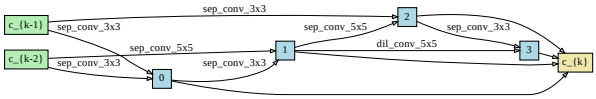


Figure 5: Normal Cell Architecture

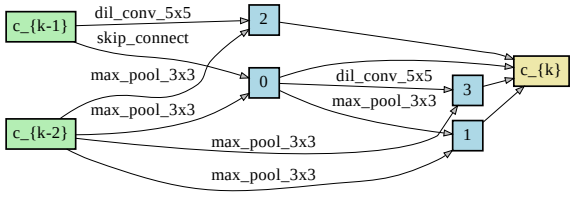


Figure 6: Reduction Cell Architecture

We report the architecture searched based on the above non-IID dataset and hyper-parameter setting. Figure 5 and 6 show the

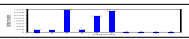
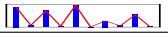
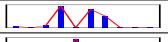
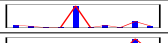
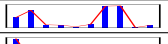
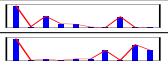
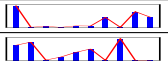
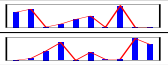
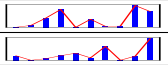
Client ID	Numbers of samples in the classes										Distribution
	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
k=0	144	94	**1561**	133	**1099**	**1466**	**0**	**0**	**0**	**0**	
k=1	327	28	264	16	354	2	100	20	200	3	
k=2	6	6	641	1	255	4	1	2	106	**1723**	
k=3	176	792	100	28	76	508	991	416	215	**0**	
k=4	84	**1926**	1	408	133	24	771	**0**	**0**	**0**	
k=5	41	46	377	541	7	235	54	**1687**	666	**0**	
k=6	134	181	505	720	123	210	44	58	663	221	
k=7	87	2	131	**1325**	**1117**	704	**0**	**0**	**0**	**0**	
k=8	178	101	5	32	**1553**	10	163	9	437	131	
k=9	94	125	**0**	147	287	100	23	217	608	279	
k=10	379	649	106	90	35	119	807	819	3	85	
k=11	**1306**	55	681	227	202	34	**0**	648	**0**	**0**	
k=12	**1045**	13	53	6	77	70	482	7	761	494	
k=13	731	883	15	161	387	552	4	**1051**	**0**	**0**	
k=14	4	97	467	899	**0**	407	50	64	**1098**	797	
k=15	264	2	93	266	412	142	806	2	243	**1267**	

Table 2: The actual heterogeneous data distribution (non-IID) used in experiments

normal cell architecture and the reduction cell architecture, respectively. We can see that the reduction cell uses more pooling operations while the normal cell has more convolutional operations.

C. Future Works

Our future work aims to improve the FedNAS framework from form the following perspectives.

- **Local NAS under Resource Constraint**. Our current search space fits for cross-organization federated learning, where the edge device can be equipped with powerful GPU devices. But when used in resource-constrained environments such as smartphones or IoT devices, the memory of our search space is too large. Searching on compact search space or using sampling methods are potential solutions to this challenge.
- **One Stage NAS**. In our FedNAS framework, we first search the architecture and then train it from scratch. This two-stage training process can be optimized by being merged into a single process. In FedNAS, a naive way to perform one-stage NAS is to continue training the model after the search process is finished. More advanced methods to further improve efficiency are open problems.
- **Asynchronous Aggregation**. The searching or training time can be reduced by asynchronous aggregation when each worker has a different number of samples, communication abilities, and computational resources. For FedNAS, the impact of the asynchronous aggregation on model performance should be evaluated.
- **Personalized NAS**. Since the data distribution in each client is heterogeneous, a personalized model in each worker may have better accuracy than a global shared model. To meet this requirement, searching for personalized architectures is a challenging task.