Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing

He Li, Kaoru Ota, and Mianxiong Dong

Abstract

Deep learning is a promising approach for extracting accurate information from raw sensor data from IoT devices deployed in complex environments. Because of its multilayer structure, deep learning is also appropriate for the edge computing environment. Therefore, in this article, we first introduce deep learning for IoTs into the edge computing environment. Since existing edge nodes have limited processing capability, we also design a novel offloading strategy to optimize the performance of IoT deep learning applications with edge computing. In the performance evaluation, we test the performance of executing multiple deep learning tasks in an edge computing environment with our strategy. The evaluation results show that our method outperforms other optimization solutions on deep learning for IoT.

Introduction

In recent years, deep learning has become an important methodology in many informatics fields such as vision recognition, natural language processing, and bioinformatics [1, 2]. Deep learning is also a strong analytic tool for huge volumes of data. In the Internet of Things (IoT), one open problem is how to reliably mine real-world IoT data from a noisy and complex environment that confuses conventional machine learning techniques. Deep learning is considered as the most promising approach to solving this problem [3]. Deep learning has been introduced into many tasks related to IoT and mobile applications with encouraging early results. For example, deep learning can precisely predict the home electricity power consumption with the data collected by smart meters, which can improve the electricity supply of the smart grid [4]. Because of its high efficiency in studying complex data, deep learning will play a very important role in future IoT services.

Edge computing is another important technology for IoT services [5–7]. Due to data transferring with limited network performance, the centralized cloud computing structure is becoming inefficient for processing and analyzing huge amounts of data collected from IoT devices [8, 9]. As edge computing offloads computing tasks from the centralized cloud to the edge near IoT devices, transferred data are enormously reduced by the preprocessing procedures. The edge computing can perform well when the intermediate data size is smaller than the input data size.

A typical deep learning model usually has many layers in the learning network. The intermediate data size can be quickly scaled down by each network layer until enough features are found. Therefore, the deep learning model is very appropriate for the edge computing environment since it is possible to offload parts of learning layers in the edge and then transfer the reduced intermediate data to the centralized cloud server.

Another advantage of deep learning in edge computing is the privacy preserving in intermediate data transferring. Intermediate data generated in traditional big data systems, such as MapReduce or Spark, contains the user privacy since the preprocessing remains as data semantics. The intermediate data in deep learning usually have different semantics compared to the source data. For example, it is very hard to understand the original information with the features extracted by a convolutional neural network (CNN) filter in the intermediate CNN layer.

Thus, in this article, we introduce deep learning for IoT into the edge computing environment to improve learning performance as well as to reduce network traffic. We formulate an elastic model that is compatible with different deep learning models. Thus, because of the different intermediate data size and preprocessing overhead of different deep learning models, we state a scheduling problem to maximize the number of deep learning tasks with the limited network bandwidth and service capability of edge nodes. We also try to guarantee the quality of service (QoS) of each deep learning service for IoT in the scheduling. We design offline and online scheduling algorithms to solve the problem. We perform extensive simulations with multiple deep learning tasks and given edge computing settings. The experimental results show that our solution outperforms other optimization methods on deep learning for IoT.

The main contributions of this article are summarized as follows. We first introduce deep learning for IoT into the edge computing environment. To the best of our knowledge, this is an innovative work focusing on deep learning for IoT with edge computing. We formulate an elastic model for varying deep learning models for IoT in edge computing. We also design an efficient online algorithm to optimize the service capacity of the edge computing model. Finally, we test the deep learning model for IoT with extensive experiments in a given edge computing environment. We also compare our edge computing method to traditional solutions.

The remainder of this article can be outlined

Digital Object Identifier:
10.1109/MNET.2018.1700202

The authors are with Muroran Insitute of Technology

0890-8044/18/$25.00 © 2018 IEEE

as follows. The next section introduces the deep learning technology for IoT and edge computing. Then we discuss the deep learning services for IoT in the edge computing environment. Following that, we describe the problem and solutions of scheduling IoT deep learning tasks in edge computing. Then we present the performance evaluation results of deep learning for IoT through extensive experiments, followed by conclusions given in the final section.

Related Work

In this section, we first introduce related technologies on deep learning for IoT and then discuss edge computing and deep learning.

Deep Learning for IoT

Deep learning is becoming an emerging technology for IoT applications and systems. The most important benefit of deep learning over machine learning is better performance with large data scale since many IoT applications generate a large amount of data for processing. Another benefit is that deep learning can extract new features automatically for different problems. In processing multimedia information, the performance of traditional machine learning depends on the accuracy of the features identified and extracted. Since it can precisely learn high-level features such as human faces in images and language words in voices, deep learning can improve the efficiency of processing multimedia information. Meanwhile, deep learning takes much less time to inference information than traditional machine learning methods.

Therefore, the development of IoT devices and technologies brings preconditions for complex deep learning tasks. Because of limited energy and computing capability, an important issue is executing deep learning applications in IoT devices. General commercial hardware and software fall short of supporting high-parallel computing in deep learning tasks. Lane *et al.* [10] proposed new acceleration engines, such as DeepEar and DeepX, to support different deep learning applications in the latest mobile systems on chips (SoCs). From the experimental results, mobile IoT devices with high-spec SoCs can support part of the learning process.

Introducing deep learning into more IoT applications is another important research issue [11]. The efficiency of deep learning for IoT has been evaluated in many important IoT applications. For example, some works focus on the applications in wearable IoT devices deployed in dynamic and complex environments that often confuse the traditional machine learning methods. Bhattacharya *et al.* [12] proposed a new deep learning model for wearable IoT devices that improves the accuracy of audio recognition tasks.

Most existing deep learning applications (e.g., speech recognition) still need to be cloud-assisted. Alsheikh *et al.* [13] proposed a framework to combine deep learning algorithms and Apache Spark for IoT data analytics. The inference phase is executed on mobile devices, while Apache Spark is deployed in cloud servers for supporting data training. This two-layer design is very similar to edge computing, which shows that it is possible to offload processing tasks from the cloud.

The most important benefit of deep learning over machine learning is the better performance with large data scale since many IoT applications generate a large amount of data for processing. Another benefit is that deep learning can extract new features automatically for different problems.

Deep Learning and Edge Computing

Edge computing is proposed to move computing ability from centralized cloud servers to edge nodes near the user end. Edge computing brings two major improvements to the existing cloud computing. The first one is that edge nodes can preprocess large amounts of data before transferring them to the central servers in the cloud. The other one is that the cloud resources are optimized by enabling edge nodes with computing ability [14]. Due to the potentiality brought by edge computing, the aforementioned problems of the cloud infrastructure can be well addressed.

Liu *et al.* [15] proposed the first work to introduce deep learning into the edge computing environment. They proposed a deep-learning-based food recognition application by employing edge-computing-based service infrastructure. Their work shows that edge computing can improve the performance of deep learning applications by reducing response time and lowering energy consumption. However, this work considered mobile phones as edge nodes, which is not appropriate for IoT services since most IoT devices are equipped only with low-spec chips. Since we focus on general IoT devices without enough energy supplement and high-spec chips, the edge servers are deployed in IoT gateways, which have enough service capacity for executing deep learning algorithms.

Deep Learning for IoT in Edge Computing

In this section, we first introduce the scenario of deep learning for IoT and then present the edge computing framework of deep learning for IoT.

Usually, IoT devices generate large amounts of data and transfer data to the cloud for further processing. These data include multimedia information, such as video, images, and sounds, or structured data, such as temperature, vibration, and luminous flux information. There are many mature technologies for processing structured data and then automatically controlling IoT devices. Traditional multimedia processing technologies, which need complex computations, are not appropriate for IoT services. Since the deep learning technology improves the efficiency of processing multimedia information, more and more works have begun to introduce deep learning into multimedia IoT services.

Video sensing is an important IoT application, which integrates image processing and computer vision in IoT networks. It is still a challenge to recognize objects from low-quality video data recorded by IoT devices. Since deep learning shows very promising accuracy in video recognition, we consider it as a typical IoT application with deep learning. Thus, as shown in Fig. 1, we use a video recognition IoT application as the example to introduce deep learning for IoT.

There are several wireless video cameras monitoring the environment and recognizing objects. The wireless cameras collect 720p video data

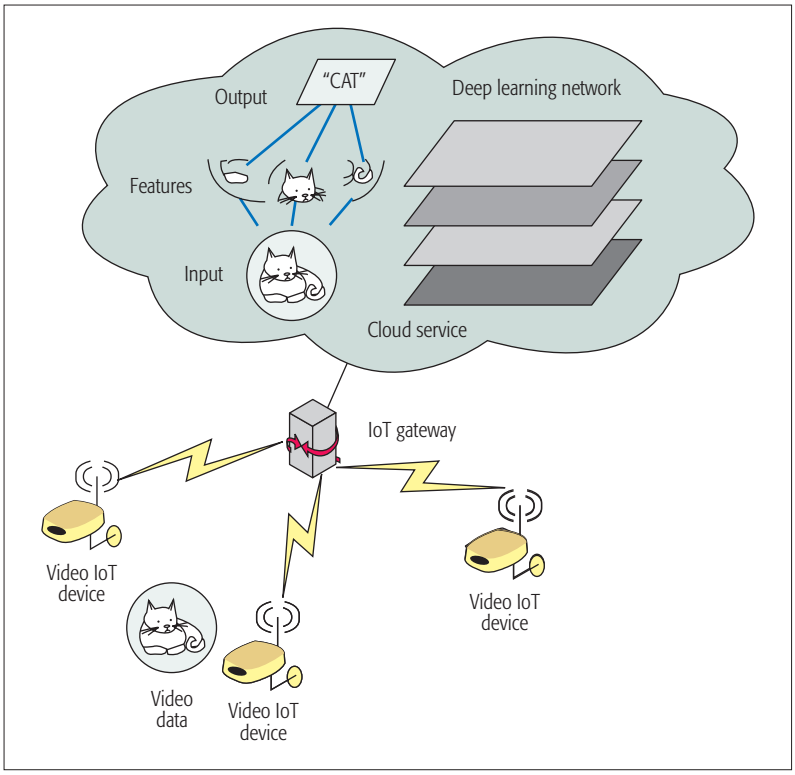

FIGURE 1. Deep learning for video recognition with IoT devices.

with a bit rate of 3000 kb/s. Then the cameras transfer the collected data to the IoT gateway through general WiFi connections. IoT gateways forward all collected data to the cloud service through Internet communications after coding and compressing the raw video data. The cloud service recognizes the objects in the collected video data through a deep learning network.

A deep learning network usually has multiple layers. The input data will be processed in these layers. Each layer processes the intermediate features generated by the previous layer and then generates new features. Finally, the extracted features generated by the last deep learning network layer will be processed by a classifier and recognized as the output. In deep learning networks, we consider the layers near input data to be lower layers; others are higher layers.

In the example, we use AlexNet to identify the object in the collected video data. AlexNet has eight layers in which the first five layers are convolutional layers, and the following three layers are fully connected layers. We first train the deep learning network with an open dataset from Kaggle, which is comprised of 25,000 dog and cat images. The deep learning application wants to detect the correct animal in the corrected video data. We use a transfer learning technique to build the classifier, which outputs the text "cat" or "dog" after processing all extracted features.

Deep learning improves the efficiency of multimedia processing for IoT services since features are extracted by multiple layers instead of traditional complex preprocessing. However, the communication performance will be the bottleneck with improved processing efficiency. The collected multimedia data size is much larger than traditional structured data size, but it is hard to improve the performance of the network for transferring collected data from IoT devices to the cloud service. In the example, each camera needs a bandwidth of 3 Mb/s for upgrading video data, while the IoT gateway needs 9 Mb/s.

Edge computing is a possible solution to the problem of transferring collected data from IoT devices to the cloud. In the IoT network, there are two layers, the edge layer and the cloud layer, for connecting IoT devices and the cloud service. The edge layer usually consists of IoT devices, an IoT gateway, and network access points in local area networks. The cloud layer includes the Internet connections and cloud servers. Edge computing means the processing is performed in the edge layer instead of the cloud layer. In the edge computing environment, since only the intermediate data or results need to be transferred from the devices to the cloud service, the pressure on the network is relieved with less transferring data.

Edge computing is very suitable for the tasks in which the size of intermediate data is smaller than the input data. Therefore, edge computing is efficient for deep learning tasks, since the size of extracted features is scaled down by the filters in deep learning network layers. In the example, the intermediate data size generated by the first layer is $134 \times 89 \times 1$ B/frame and 2300 kb/s if we want to recognize each frame. If we only want to process keyframes in the video data, the size of the generated intermediate data is only 95 kb/s.

As shown in Fig. 2, we present an edge computing structure for IoT deep learning tasks. The structure consists of two layers as well as a typical edge computing structure. In the edge layer, edge servers are deployed in IoT gateways for processing collected data. We first train the deep learning networks in the cloud server. After the training phase, we divide the learning networks into two parts. One part includes the lower layers near the input data, while another part includes the higher layers near the output data.

We deploy the part with lower layers into edge servers and the part with higher layers into the cloud for offloading processing. Thus, the collected data are input into the first layer in the edge servers. The edge servers load the intermediate data from the lower layers and then transferred data to the cloud server as the input data for the higher layers. In the example, if we deploy the first layer in the IoT gateway, the intermediate data with the size of $134 \times 89 \times 1$ B/frame will be sent to the second layer in the cloud server for further processing.

A problem is how to divide each deep learning network. Usually, the size of the intermediate data generated by the higher layers is smaller than that generated by the lower layers. Deploying more layers into edge servers can reduce more network traffic. However, the server capacity of edge servers is limited compared to cloud servers. It is impossible to process infinite tasks in edge servers. Every layer in a deep learning network will bring additional computational overhead to the server. We can only deploy part of the deep learning network into edge servers. Meanwhile, as different deep learning networks and tasks have different sizes of intermediate data and computational overhead, efficient scheduling is needed to

optimize deep learning for IoT in the edge computing structure. We design an efficient scheduling strategy for this problem and discuss it in the next section.

Scheduling Problem and Solution

In this section, we first state the scheduling problem in the edge computing structure for IoT deep learning and then present the solution.

In a given edge computing environment, we use a set E to denote all edge servers and e_i to denote an edge server in set E. From edge server e_i to the cloud server, we use a value c_i to denote the service capacity and b_i to denote the network bandwidth. We also add a threshold value denoted by V to avoid network congestion since there is some interaction traffic between the edge servers and the cloud servers. Thus, the maximum available bandwidth between edge server e_i and the cloud server is denoted by $b_i \cdot V$.

Let set T denote all deep learning tasks and t_j denote a deep learning task in set T. The number of task t_j's deep learning network layers is N_j. We assume the reduced data size is near an average value for each task with different input data. The average ratio of the intermediate data size generated by the kth layer ($k \in [1, N_j]$) to the total input data size is denoted by r_{kj}. For task t_j and edge server e_i, assigned bandwidth is denoted by b_{ij}. Let d_{ij} denote the input data size per time unit of task t_j in edge server e_i. Thus, the transferring latency of task t_j in edge server e_i can be denoted $d_{ij} \cdot r_{kj}/b_{ij}$, if k layers of task t_j are placed in edge server e_i. For guaranteeing QoS, the transferring latency should be smaller than a maximum value denoted by Q_j. For task t_j, the computational overhead of a unit of input data after the kth layer is denoted by l_{kj}. Therefore, for task t_j, the computational overhead in edge server e_i is $l_{kj} \cdot d_{ij}$.

The Problem of Scheduling IoT Deep Learning Network Layers in Edge Computing: Given an edge computing structure, the scheduling problem attempts to assign maximum tasks in the edge computing structure by deploying deep learning layers in IoT edge servers such that the required transferring latency of each task can be guaranteed, denoted by

$$\begin{aligned} \max \quad & \sum_{i=1}^{|E|}\sum_{j=1}^{|T|} X_{ij} \\ s.t., \quad & \sum_{i=1}^{|E|} b_{ij} \le b_i \cdot V \\ & X_{ij} \cdot d_{ij} \cdot r_{kj}/b_{ij} \le Q_j \\ & \sum_{j=1}^{|T|} l_{kj} \cdot d_{ij} \cdot X_{ij} \le c_i \end{aligned}$$

where $X_{ij} = 1$ if task t_j is deployed in edge server e_i; otherwise, $X_{ij} = 0$.

We propose an offline algorithm and an online algorithm to solve the scheduling problem. The offline scheduling algorithm first finds out k_j^m, which maximizes the value of $r_{kj} \cdot l_{kj}$, and edge server i_j^m, which has the largest input data of task t_j. Then the algorithm sorts all tasks in ascending order of the largest input data size. The scheduling first deploys task t_j with minimum input data size to edge servers. The algorithm traverses all edge servers to check whether an edge server has enough service capability and network bandwidth to deploy task t_j. If all edge servers have enough service capacity and bandwidth, the algorithm deploys task t_j into all edge servers. If an edge server does not have enough uploading bandwidth or service capacity, the algorithm changes the value of k and find out an appropriate k for deploying task t_j in all edge servers. If the edge server does not have enough service capacity or network bandwidth even after varying k, the scheduling algorithm will not deploy task t_j in edge servers.

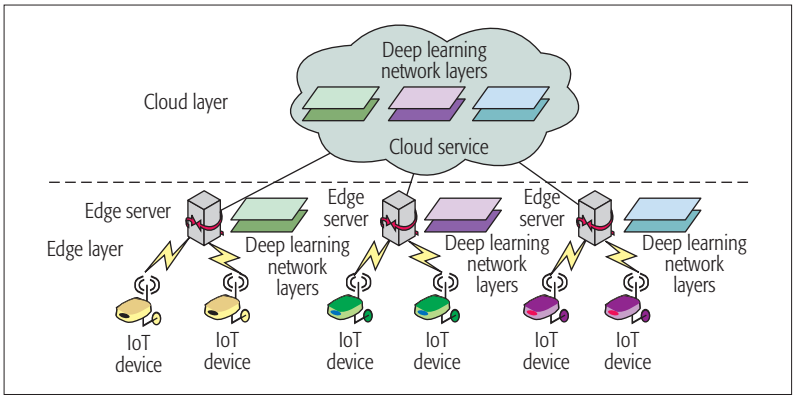

FIGURE 2. Edge computing structure for IoT deep learning.

In the worst case, the complexity of the offline algorithm is $\mathcal{O}(|T| \cdot |E|^2 \cdot K)$ where K is the maximum number of deep learning network layers of each task. Since the number of tasks is much larger than the number of edge servers and deep learning network layers, the complexity of the proposed algorithm is $\mathcal{O}(|T|)$, which is good enough for practical scheduling. We also analyze the efficiency of the algorithm and find that the approximate ratio is $2/V$.

Meanwhile, we design an online scheduling algorithm that decides the deployment when task t_j is coming. As the task scheduling has little information about feature tasks, the deployment decision is based on the historical tasks. We use B^{max} and B^{min} to denote the maximum and minimum required bandwidth of a task, respectively. Thus, for task t_j, we first calculate the k_j^m and i_j^m. Then we define a value $\Phi(c_{i_j^m}) \leftarrow (B^{min} \cdot e/B^{max})^{c_{i_j^m}} \cdot (B^{max} \cdot e)$, where the remaining service capacity of edge server $e_{i_j^m}$ is $c_{i_j^m}$ and e is the mathematical constant. If $(b_{i_j^m} - d_{i_j^m j} \cdot r_{i_j^m j}/Q_j)) \cdot (c_{i_j^m} - d_{i_j^m j} \cdot l_{i_j^m j}) \le \Phi(c_{i_j^m})$ and other edge servers have enough bandwidth and service capacity, the scheduling algorithm deploys task t_j into edge servers. The approximate ratio of the online algorithm is

$$\frac{1}{\left(\ln\left(B^{max}/B^{min}\right)+1\right)\cdot V}.$$

Performance Evaluation

In this section, we first describe the experiment settings and then discuss the performance evaluation result.

In the experiments, we have two environments, one for collecting data from deep learning tasks and another for simulations. For executing deep learning applications, we use a workstation equipped with an Intel Core i7 7770 CPU and NVIDIA Geforce GTX 1080 graphic card. We use Caffe as the CNN framework and define 10 different CNN networks. We execute 10 CNN tasks

A deep learning network usually has multiple layers. The input data will be processed in these layers. Each layer processes the intermediate features generated by the previous layer and then generates new features. Finally, the extracted features generated by the last deep learning network layer will be processed by a classifier and recognized as the output.

with different CNN networks and record the number of operations and intermediate data generated in each CNN layer.

As shown in Fig. 3, we choose two deep learning networks, CNN1 and CNN2, as the example for illustration of the reduced data size ratio (blue plots) and computational overhead (red plots). These two deep learning networks have five layers and different neuron settings. From the plots, the input data can be reduced by the deep learning networks, and more intermediate data are reduced by lower layers. Meanwhile, the computational overhead is increased quickly with more layers.

We use Python 2.7 and networkx to develop the simulator and use the reduced ratio of the intermediate data from executing CNN tasks. In the simulations, we set the number of deep learning tasks to 1000. The service capability of each edge server is set to 290 Gflops according to NVIDIA Tegra K1. We set the number of edge servers in the network from 20 to 90. The input data size of each task is set from 100 kB to 1 MB. The layer number of all CNN networks is set from 5 to 10. The bandwidth of each edge server is uniformly distributed from 10 Mb/s to 1 Gb/s. The required latency is set to 0.2 s.

We first test the performance of the layer scheduling algorithm. We set the number of edge servers from 20 to 90 and increase the number by 10 in each step. We compare the performance with the fixed mode that deploys a fixed number of deep learning layers in edge servers. We set the number of deep learning layers in the fixed mode from 1 to 5. As shown in Fig. 4a, the scheduling algorithm outperforms the fixed mode with a different number of layers. Meanwhile, with more edge servers, more deep learning tasks can be deployed in the network. We find that the fixed mode with two layers deployed in edge servers performs better than other settings. For most deep learning networks in our simulation, deploying two layers in the edge servers can leverage the computational overhead and uploading bandwidth.

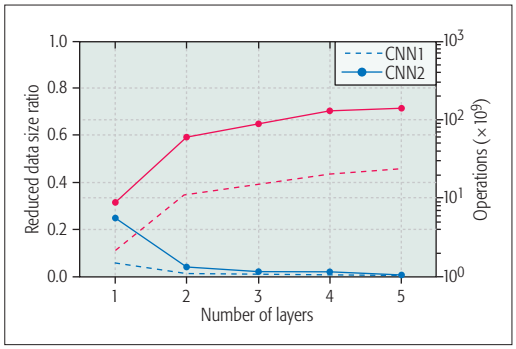

FIGURE 3. Reduced data and operations in deep learning networks.

Then we test the performance of the online algorithm. We also compare the performance of the online algorithm with two popular online scheduling algorithms, first in first out (FIFO) and low bandwidth first deployment (LBF). We input a random sequence of 1000 tasks to the edge network, and these two algorithms deploy tasks into edge servers. The number of edge servers is set to 50. As shown in Fig. 4b, the FIFO algorithm deploys every task until there is not enough capability and bandwidth. Thus, after deploying 360 tasks, the FIFO algorithm pops out the first deployed tasks for appending the following tasks. LBF is similar to FIFO when the capacity and bandwidth are enough. When there is no space for deploying the following tasks, LBF removes the task with maximum bandwidth requirement. Our online algorithm will decide whether the following task should be deployed into edge servers. Thus, when the number of input tasks is near 600, the online algorithm deploys more tasks than FIFO. When the number of input tasks is near 800, the online algorithm deploys more tasks than LBF. As a result, our algorithm outperforms FIFO and LBF algorithms over a long time period.

Conclusion and Future Work

In this article, we introduce deep learning for IoT into the edge computing environment to optimize network performance and protect user privacy in uploading data. The edge computing structure

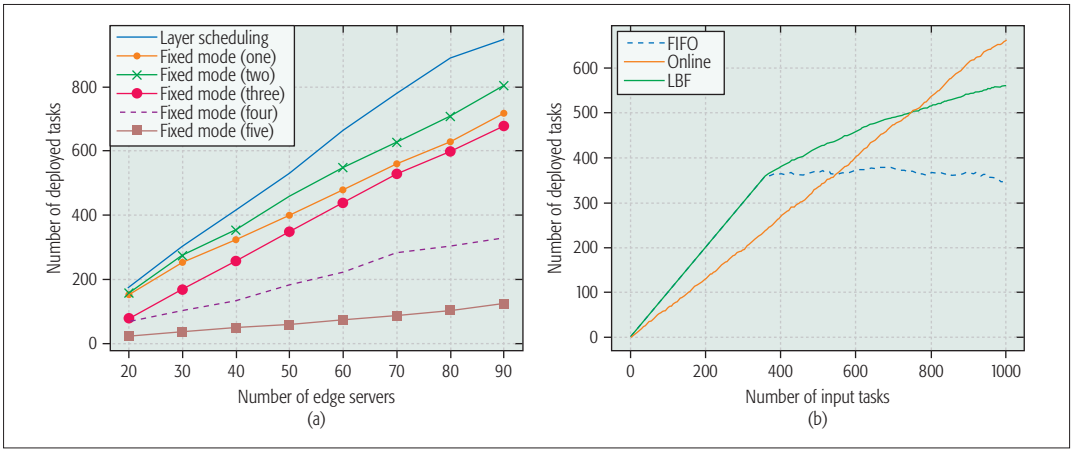

FIGURE 4. Number of deployed tasks with different algorithms: a) layer sheduling and fixed mode with different number of edge servers; b) online algorithm and FIFO algorithm.

reduces the network traffic from IoT devices to cloud servers since edge nodes upload reduced intermediate data instead of input data. We also consider the limited service capability of edge nodes and propose algorithms to maximize the number of tasks in the edge computing environment. In the experiments, we choose 10 different CNN models as the deep learning networks and collect the intermediate data size and computational overhead from practical deep learning applications. The results of the performance evaluation show that our solutions can increase the number of tasks deployed in edge servers with guaranteed QoS requirements. As future work, we plan to deploy deep learning applications in a real-world edge computing environment with our algorithms.

The results of the performance evaluation show that our solutions can increase the number of tasks deployed in edge servers with guaranteed QoS requirements. As future work, we plan to deploy deep learning applications in a real-world edge computing environment with our algorithms.

Acknowledgments

This work is supported by JSPS KAKENHI Grant Numbers JP16K00117, JP15K15976, and JP17K12669, the KDDI Foundation, and the Research Fund for Postdoctoral Program of Muroran Institute of Technology. Mianxiong Dong is the corresponding author.

References

[1] Z. Fadlullah *et al.*, "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems," *IEEE Commun. Surveys & Tutorials*, DOI: 10.1109/COMST.2017.2707140.

[2] N. Kato *et al.*, "The Deep Learning Vision for Heterogeneous Network Traffic Control: Proposal, Challenges, and Future Perspective," *IEEE Wireless Commun.*, vol. 24, no. 3, June 2017, pp. 146–53. DOI: 10.1109/MWC.2016.1600317WC.

[3] S. Verma *et al.*, "A Survey on Network Methodologies for Real-Time Analytics of Massive Iot Data and Open Research Issues," *IEEE Commun. Surveys & Tutorials*. DOI: 10.1109/COMST.2017.2694469.

[4] L. Li, K. Ota, and M. Dong, "When Weather Matters: IoT-Based Electrical Load Forecasting for Smart Grid," *IEEE Commun. Mag.*, vol. 55, no. 10, Oct. 2017, pp. 46–51.

[5] T. G. Rodrigues *et al.*, "Hybrid Method for Minimizing Service Delay in Edge Cloud Computing through VM Migration and Transmission Power Control," *IEEE Trans. Computers*, vol. 66, no. 5, May 2017, pp. 810–19.

[6] Y. Zhang *et al.*, "A Survey on Emerging Computing Paradigms for Big Data," *Chinese J. Electronics*, vol. 26, no. 1, 2017, pp. 1–12.

[7] J. Ren *et al.*, "Serving at the Edge: A Scalable IoT Architecture Based on Transparent Computing," *IEEE Network*, 2017.

[8] J. Liu *et al.*, "Energy Consumption Minimization for FiWi Enhanced Lte-A Hetnets with UE Connection Constraint," *IEEE Commun. Mag.*, vol. 54, no. 11, Nov. 2016, pp. 56–62.

[9] J. Liu *et al.*, "New Perspectives on Future Smart FiWi Networks: Scalability, Reliability, and Energy Efficiency," *IEEE Commun. Surveys & Tutorials*, vol. 18, no. 2, 2nd qtr. 2016, pp. 1045–72.

[10] N. D. Lane, P. Georgiev, and L. Qendro, "Deepear: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning," *Proc. 2015 ACM Int'l. Joint Conf. Pervasive and Ubiquitous Computing*, 2015, pp. 283–94.

[11] L. Li *et al.*, "Eyes in the Dark: Distributed Scene Understanding for Disaster Management," *IEEE Trans. Parallel Distrib. Systems*, 2017. DOI: 10.1109/TPDS.2017.2740294.

[12] S. Bhattacharya and N. D. Lane, "Sparsification and Separation of Deep Learning Layers for Constrained Resource Inference on Wearables," *Proc. 14th ACM Conf. Embedded Network Sensor Systems CD-ROM, ser. SenSys '16*, 2016, pp. 176–89.

[13] M. A. Alsheikh *et al.*, "Mobile Big Data Analytics Using Deep Learning and Apache Spark," *IEEE Network*, vol. 30, no. 3, May/June 2016, pp. 22–29.

[14] T. X. Tran *et al.*, "Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, Apr. 2017, pp. 54–61.

[15] C. Liu *et al.*, "A New Deep Learning-Based Food Recognition System for Dietary Assessment on an Edge Computing Service Infrastructure," *IEEE Trans. Services Computing*. DOI: 10.1109/TSC.2017.2662008.

Biographies

He Li received his B.S., and M.S. degrees in computer science and engineering from Huazhong University of Science and Technology in 2007 and 2009, respectively, and his Ph.D. degree in computer science and engineering from the University of Aizu, Japan, in 2015. He is currently a postdoctoral fellow with the Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan. His research interests include cloud computing and software defined networking.

Kaoru Ota received her M.S. degree in computer science from Oklahoma State University in 2008, and her B.S. and Ph.D. degrees in computer science and engineering from the University of Aizu in 2006 and 2012, respectively. She is currently an assistant professor with the Department of Information and Electronic Engineering, Muroran Institute of Technology. She serves as an Editor for *IEEE Communications Letters*.

Mianxiong Dong received his B.S., M.S., and Ph.D. in computer science and engineering from the University of Aizu. He is currently an associate professor in the Department of Information and Electronic Engineering at Muroran Institute of Technology, Japan. He serves as an Editor for *IEEE Communications Surveys & Tutorials*, *IEEE Network*, *IEEE Wireless Communications Letters*, *IEEE Cloud Computing*, and *IEEE Access*.