Xu et al. EURASIP Journal on Wireless Communications and Networking (2020) 2020:211
https://doi.org/10.1186/s13638-020-01794-2

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

A collaborative cloud-edge computing framework in distributed neural network

Shihao Xu[1], Zhenjiang Zhang[1,2*], Michel Kadoch[3] and Mohamed Cheriet[3]

* Correspondence: zhangzhenjiang@bjtu.edu.cn
[1]School of Electronic and Information Engineering, Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University, Beijing, China
[2]School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China
Full list of author information is available at the end of the article

Abstract

The emergence of edge computing provides a new solution to big data processing in the Internet of Things (IoT) environment. By combining edge computing with deep neural network, it can make better use of the advantages of multi-layer architecture of the network. However, the current task offloading and scheduling frameworks for edge computing are not well applicable to neural network training tasks. In this paper, we propose a task model offloading algorithm by considering how to optimally deploy neural network model into the edge nodes. An adaptive task scheduling algorithm is also designed to adaptively optimize the task assignment by using the improved ant colony algorithm. Based on them, a collaborative cloud-edge computing framework is proposed, which can be used in the distributed neural network. Moreover, this framework sets up some mechanisms so that the cloud can collaborate with edge computing in the work. The simulation results show that the framework can reduce time delay and energy consumption, and improve task accuracy.

Keywords: Edge computing, Distributed neural network, Resource allocation, Task offloading

1 Introduction

With the rapid development of the Internet of Things (IoT), cloud computing provides enabling technologies for the storage and processing of sensor data [1], especially in the industrial circle [2]. However, cloud computing leads to high latency and requires high transmission bandwidth. In addition, the cloud is usually unreliable [3]. Edge computing [4], which has emerged as a new calculation paradigm, can solve these problems. Since the edge nodes are usually closer to the sensors than the remote cloud [5], edge computing can reduce the latency and bandwidth, and it is safer than cloud computing.

Recently, edge computing has been widely used for computationally intensive tasks of artificial intelligence (AI) in the actual IoT environment [6]. Since the edge nodes have the feature of resource-constrained and dynamic changes, it is significant to design an appropriate framework to offload and schedule computational tasks in the edge. Zhang et al. in [7] proposed a multiple algorithm service model (MASM) to offload AI tasks to the cloudlet server and designed an energy-delay optimization model specifically for the edge AI scenario. In [8], Wang et al. considered the problem of

learning model parameters from the data that distributed among multiple edge nodes and they proposed an adaptive federated learning framework. On the hardware level, Li et al. in [9] presented an architectural design and implementation of a natural scene text interpretation accelerator, which reduced the communication overhead from the edge device to the server. There are also some works which considered multi-task priority edge computing [10] and maximized the profit of mobile network operator by jointly optimizing service rate, transmit power, and subcarrier allocation with satisfying power and delay constraints, which takes full advantage of edge node resources.

Deep neural network (DNN), also known as deep learning, is suitable to handle the IoT tasks, because it learns feature automatically from the big data. DNN has been applied to many aspects of IoT, such as intelligent monitoring [11], smart home [12], and industrial manufacture [13]. It can be imagined that if DNN can be deployed in the edge distributedly, it will help to resolve computationally intensive tasks in the IoT. Thus, it requires a suitable solution to offload and schedule tasks on the resource-constrained edge nodes. For example, the fast-convolutional neural networks (fast-CNN) [14] is widely used in intelligent monitoring. Since the calculation of each convolution layer is independent, it is feasible to execute part layers of network separately on the edge and the cloud [15].

[6–8] designed edge computing frameworks to handle AI tasks, but they were only available when the resource of restricted IoT edge devices strictly meets the requirements of computation. Surat et al. in [15] proposed distributed deep neural networks to limit use of edge nodes. Due to its distributed nature, the architecture could improve fault tolerance for application execution within a sensing fusion system. Based on that work, the encoding of feature space was proposed in [16] to enhance the maximum input rate supported by the edge platform and reduce the energy of the edge platform. In [17], Zhu et al. proposed a literal multi-dimensional anomaly detection approach using the distributed long-short-term-memory (LSTM) framework for the in-vehicle network. To enhance the accuracy and efficiency of detection, it detected anomaly from both time and data dimension simultaneously by exploiting multi-task LSTM neural network on mobile edge. Zhao et al. in [18] proposed the DeepThings framework, which used a scalable converged tile partition convolutional layer to minimize memory footprint and further implements distributed work-stealing methods to assign workload dynamically in an inferential runtime environment of IoT.

These existing systems provide good ideas of deploying deep neural network into edge nodes, but they do not consider the optimal offloading of task models and load-balancing. In [19], Kang et al. designed a Neurosurgeon framework, which adapted to various DNN architectures, hardware platforms, wireless networks, and server load levels, intelligently offloading computation model. Many attempts have been made to optimally balance the workload. Xiao et al. in [20] proposed a collaborative load-balancing algorithm for the TS mechanism in edge computing nodes and achieved Pareto optimality through the collaborative working to improve the performance of every node [21]. presented a work-sharing model Honeybee, using an adaptation of the well-known work- stealing method to load independent balance jobs among heterogeneous mobile nodes, able to accommodate nodes randomly leaving and joining the system [22]. studied a task scheduling problem to balance this tradeoff under cloud-edge architecture for reducing weighted transmission time, which considered learning accuracy.

There are other network resources that can also be evaluated [23]; jointly obtain subcarrier and transmit power allocation policies for both the uplink and downlink transmissions with task scheduling and computation resource allocation at both the users and the MEC server.

This paper presents a collaborative cloud-edge computing framework in distributed neural network to handle the computationally intensive tasks of neural networks in the IoT environment. A task model offloading algorithm (TMOA) is designed to configure edge nodes with neural networks by analyzing the computational intensity and time latency through the roofline model and the task arrival model and using the Lagrange multiplier method to optimize the layering offload with multiple constraints of latency, energy consumption, and process capability. An adaptive task scheduling algorithm (ATSA) is also designed for load-balancing of edge nodes, through an improved heuristic ant colony algorithm. A cloud dormancy mechanism and a parameter aggregation scheme are established to coordinate cloud-edge computing adaptively and optimize task model. This collaborative cloud-edge computing framework in distributed neural network can balance workload and reduce latency, and it has less time latency, less energy consumption, and higher accuracy than existing other frameworks.

The rest of this paper is organized as follows: Section 2 discusses the architecture of this system model and analyses the ATSA algorithm, the cloud dormancy mechanism, and the parameter aggregation scheme. Section 3 elaborates TMOA algorithm in details, and Section 4 discusses the results of the simulation. Finally, Section 5 concludes this paper.

2 System model and analysis

In this section, we firstly propose the holistic framework. Then, we illustrate the main parts of the framework. We focus on task-scheduling and collaborative cloud-edge learning. The main algorithm for edge node configuration will be proposed in section III.

2.1 Cloud-edge framework

The workflow of the framework is shown in Fig. 1. When the cloud receives tasks from the user, it completes edge node configuration of the distributed neural network through task model offloading algorithm and offloads task model to the master node of the edge side.

Next, the master node distributes the network model to the slave nodes and assigns tasks adaptively according to the adaptive task scheduling algorithm. Meanwhile, the slave nodes obtain the task data from terminal devices.

Finally, the collaborative cloud-edge learning is carried out according to the parameter aggregation scheme, and the cloud dormancy mechanism is executed dynamically to process the neural network tasks.

2.2 Edge node configuration

In order to reduce the processing load of the cloud and to make full use of the resource-constrained edge nodes, we propose the edge node configuration of the neural network architecture.

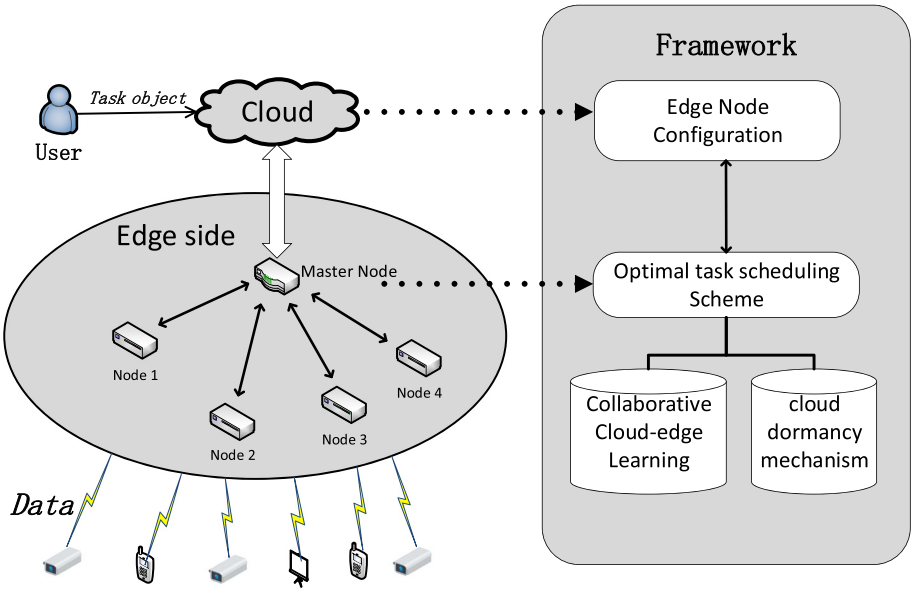

Fig. 1 The collaborative cloud-edge computing framework. The main parts of our framework, including cloud side, edge side, and device side

First, the user releases task information to the cloud, including task requirement and neural network structure. The cloud communicates with the edge side, and the available edge nodes are configured to form a resource pool. One master node is set randomly in the edge resource pool, and other nodes are set as slave nodes. Assuming that there are x slave nodes on the edge side, numbered separately as $V = \{v_1, v_2, \ldots, v_x\}$ and a master node numbered as v_0. The cloud obtains the calculation speed of X slave nodes respectively, which is measured by the maximum number of floating-point operations per second (FLOPS), denoted as $F = \{Fv_1, Fv_2, \ldots, Fv_x\}$; it obtains X slave nodes' maximum energy consumption, denoted as $E = \{Ev_1, Ev_2, \ldots, Ev_x\}$; it obtains X slave nodes' memory space, denoted as $P = \{Pv_1, Pv_2, \ldots, Pv_x\}$; and it obtains the throughput of per edge device, denoted as $L = \{Lv_1, Lv_2, \ldots, Lv_x\}$.

Second, according to the IoT task requirements, the cloud designs what task model to be deployed in edge nodes. Taking convolutional neural network (CNN) as an example, we assume the task model is an N-layers neural network. Then, we distribute the CNN into two parts for deployment, as shown in Fig. 2. The first j layers from data input to the middle layer are deployed in the edge nodes, and the(N-j) layers from the middle layer to data output are deployed in the remote cloud. Therefore, we design a task model offloading algorithm (details in Section 3) to determine the optimal offloading position between the edge and cloud side, according to the constraints on processing capability, task latency, and energy consumption of the edge nodes.

Finally, the cloud offloads the task model to the master node. The master node distributes the model to each slave node in a multicast manner. Then the edge nodes wait for the transmission of task data.

2.3 Optimal task scheduling scheme

Data tasks will be assigned to every available node to execute. In order to reduce the total task execution time, meanwhile maintain load-balancing of each edge node, it is

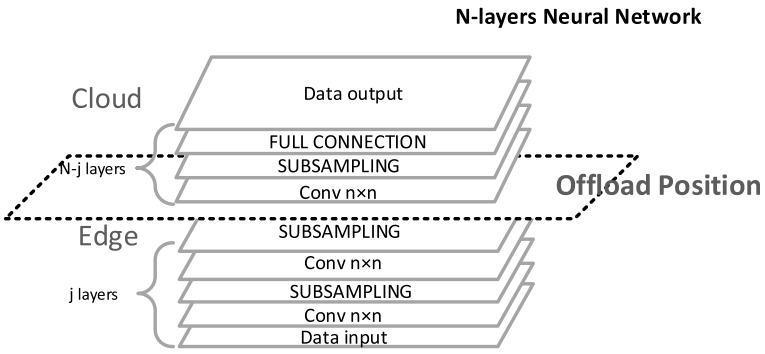

Fig. 2 Cloud-edge offloading schematic. The neural network architecture that is divided into two parts for deployment at the edge and the cloud as an example

necessary to design an efficient task scheduling algorithm. The traditional method is the first-come-first-served service (FCFS) [24]. Because it has a less time complexity, it can reduce the scheduling time. However, for the resource-limited edge nodes, the performance difference between different nodes is enormous. So, the FCFS mechanism may not achieve optimal scheduling. In this part, we design a heuristic intelligent algorithm based on ant colony optimization (ACO) [25] to optimally schedule tasks on edge nodes.

Since tasks arrive dynamically in real time, a sliding window is set to adaptively process tasks over the past period of time. Figure 3 shows the adaptive task scheduling algorithm. We assume that the tasks satisfy the following conditions: (1) the tasks are simply data intensive tasks, (2) there is no dependency between tasks, and (3) resources are exclusive by one task in one time but not shared. If the sliding window is set as K tasks $A = \{a_1, a_2, \ldots, a_k\}$ that arrive in a period time, a weight parameter $w_{v_j}^{a_i}$ is set whereas task a_i executed on node v_j, then $w_{v_j}^{a_i} = 1$; else $w_{v_j}^{a_i} = 0$. Besides, the execution time et_{ij} of learning task a_i on node v_j can be calculated in advance, denoted as $et_{ij} = \frac{Zt^j}{F_i}$, whereas F_i is the FLOP of edge node v_j and Zt^j is the offloaded models' calcula-

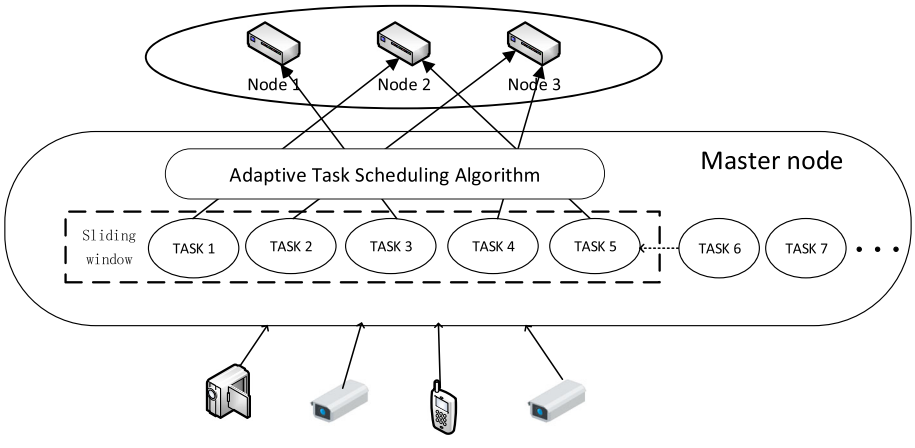

Fig. 3 Task scheduling schematic. How the adaptive task scheduling algorithm optimally schedules tasks on edge nodes

tion amount. So, all the execution time of each task on each node can be expressed in matrix form as follows:

$$ET_{ij} = \begin{bmatrix} et_{11} & et_{12} & et_{13} & \cdots & et_{1x} \\ et_{21} & et_{22} & et_{23} & \cdots & et_{2x} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ et_{k1} & et_{k2} & et_{k3} & \cdots & et_{kx} \end{bmatrix} \tag{1}$$

where et_{ij} indicates the execution time of learning task a_i on node v_j.

From $w_{v_j}^{a_i}$ and matrix ET_{ij}, we can know the total task delay of node v_j, expressed as $td_{v_j} = \sum_{i=1}^{k} w_{v_j}^{a_i} et_{ij}$. We define the path selection probability function as

$$P_{v_j}^{a_i} = \frac{\left[\tau_{i,j}\right]^{\alpha}\left[\eta_{i,j}\right]^{\beta}}{\sum_{v_j \in V}\left[\tau_{i,j}\right]^{\alpha}\left[\eta_{i,j}\right]^{\beta}} \ (1 \le \alpha, \beta \le 10) \tag{2}$$

where $\tau_{i,\,j}$ denotes the pheromone of task a_i assigned to v_j node, which can be seen as the trend to allocate task a_i to node v_j. $\eta_{i,\,j}$ denotes the heuristic information defined in relation (4), which basic principle is to balance the workload meanwhile to make full use of each node. And α and β represent the weights of pheromone and heuristic information on ant routing.

To adaptively schedule the task path, while one ant $n_y \in N = \{n_1, n_2, \ldots, n_m\}$ is looking for the optimal path, the path selection probability function will be updated as the pheromone concentration change according to the external natural environment. The change formula for the iterations:

$$\begin{cases} \tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) + \Delta\tau_{i,j}(t) \\ \Delta\tau_{i,j}(t) = \sum_{y=1}^{k} \Delta\tau_{i,j}^{y}(t) \end{cases} (0 < \rho < 1) \tag{3}$$

where ρ represents the volatility coefficient of the pheromone, $\tau_{i,\,j}(t)$ is the current pheromone that initialized to be the reciprocal of the average execution time of the processor, $\Delta\tau_{i,j}^{y}(t)$ is the pheromone of ant n_y for task a_i assigned to v_j node, , $\Delta\tau_{i,\,j}(t)$ represents the sum of pheromone of all ants that assign the task a_i to the node v_j in one iteration. We calculate $\Delta\tau_{i,j}^{y}(t) = \frac{Q}{Z_y}$, where $Z_y = \sum_{j=1}^{x}\sum_{i=1}^{k} w_{v_j}^{a_i} et_{ij}$ is the total time used by ant n_y in one iteration, and Q is the pheromone increment constant.

For the heuristic information $\eta_{i,\,j}$, it is mainly related with the memory usage percent of the node v_j, defined as μ_{v_j}.

$$\begin{cases} \mu_{v_j}(t+1) = \mu_{v_j}(t) + \dfrac{P_{cur}}{Pv_j} \\ \eta_{i,j} = \delta\left(1 - \mu_{v_j}\right) \end{cases} \tag{4}$$

where P_{cur} indicates the memory required by the current task, Pv_j indicates the total memory available in the node v_j, and δ represents the weight of the information.

According to the above relations, we designed an adaptive task scheduling algorithm as algorithm 1.

Algorithm 1 Adaptive Task Scheduling Algorithm

```
Begin
    initialize τ_{i,j}, η_{i,j}, P_{v_j}^{a_i}
    for a in iteration y
        for n_y in ants N
            put ants in a random node v_j from V
            while a_i in tasks A
                choose next node by P_{v_j}^{a_i}
                update local pheromone
            end while
        end for
        calculate best scheduling
        update global pheromone
    end for
End
```

2.4 Collaborative cloud-edge learning scheme

In our framework, there are two different schemes to offload a task model onto the edge nodes. One scheme is to deploy the pre-trained model directly to the edge nodes according to the task model offloading algorithm. Another scheme is by real-time training of the task model through collaborative cloud-edge learning.

For the second scheme, the cloud transmits the initialization parameters of the network model to the edge nodes. When the training tasks come, the node executes the task model and delivers the j th layer output feature Map_i^j to the cloud. The cloud executes the remaining part of the model and calculates the loss function. The model is trained through back-propagation, and the parameters are jointly optimized by aggregation [8], as shown in Fig. 4.

We define the loss function as $F(w)$. Each node i has a local model parameter $w_i(t)$, where $t = 0, 1, 2, \ldots$ denotes the iteration index. At $t = 0$, the local parameters for all nodes are initialized to the same value. The local model parameter is updated once every iteration, denoted as

$$\mathrm{w}_i(t) = \mathrm{w}_i(t-1) - \xi \cdot \nabla \mathrm{F}(\mathrm{w}_i(t-1)) \tag{5}$$

where $\xi > 0$ is the step size, $\nabla \mathrm{F}(w)$ is the gradient value.

And after every unit time, the model parameter of all nodes is subject to a global update, aggregate to $w(t)$, denoted as

$$w(t) = \frac{\sum_{i=1}^{u} U_i \mathrm{w}_i(t)}{\sum_{i=1}^{u} U_i} \tag{6}$$

where U_i is the number of tasks performed by node i per unit time. This parameter aggregation scheme can maintain the parameter synchronization of every edge node.

Besides, we also establish a cloud dormancy mechanism to coordinate cloud-edge computing adaptively, as shown in Fig. 4. Due to the wide distribution of edge nodes,

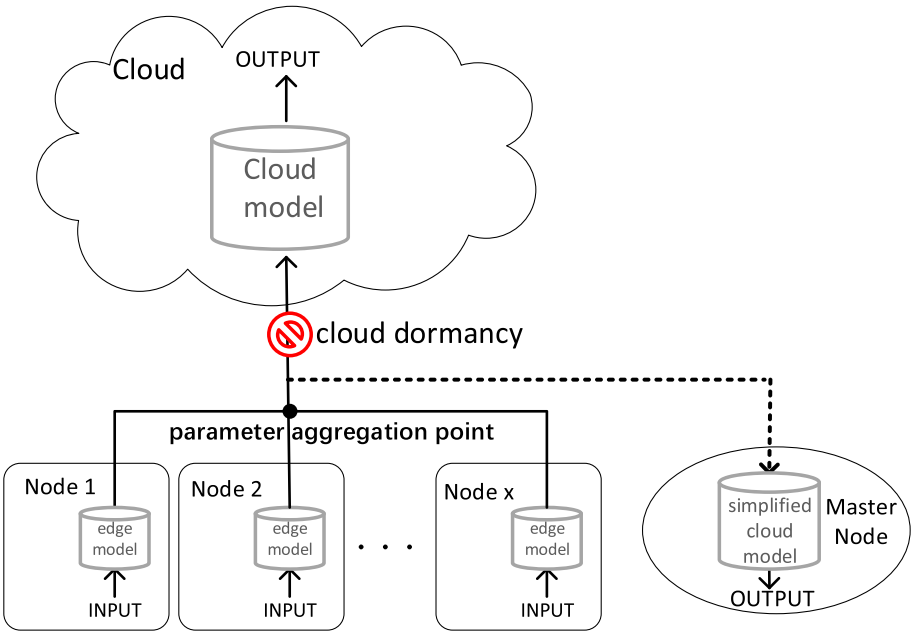

Fig. 4 Collaborative cloud-edge learning schematic. The cloud dormancy mechanism and the parameter aggregation scheme to optimize the task model

they are usually far away from the cloud, and the network link quality is hard to guarantee. During a large degree of tasks, the cloud may not meet the latency requirement. On the master node, we set a simplified cloud model consisted of a neural network [26]. If the cloud-edge communication latency cannot meet the real-time task requirements, the task data will not be uploaded to the cloud but be calculated by the simplified model on the edge. And we optimize the parameters of the edge side and the cloud jointly. The total loss function is denoted as

$$F(w) = a \cdot F_e(w) + b \cdot F_c(w) \tag{7}$$

We assume that the real-time delay from the edge to the cloud is T_{rd}.To judge the task in the edge side adaptively, we set the following rule, if the delay is less than the average delay T_0, that is, $T_{rd} \leq T_0$, then the cloud-edge collaboratively compute according to the original step; more than delay, that is, $T_{rd} > T_0$, the cloud go dormant, and the task will be computed totally on the edge side through the master node's simplified cloud model. This mechanism is in exchange for the robustness of the entire system at a slight cost of accuracy and computational complexity on edge nodes.

3 Task model offloading algorithm

Above we have illustrated the main functional modules of the framework. In this section, we will present the core algorithm of edge node configuration that determines the optimal offloading position between the edge and cloud side. In our algorithm, we analyze the computational intensity of the DNN through the roofline model and build a task arrival model to calculate the latency. We use the Lagrange multiplier method to optimize the layering offload with multiple constraints on latency, energy consumption, and process capability.

3.1 Processing capability constraint

About the constraint on processing capability, it is mainly related to the memory footprint. We use the roofline model [27] to convert the memory footprint into the spatial complexity of the model.

The different offloading positions of the task model determine the size of the memory footprint. In the neural network model, each layer performs numerous operations and occupies a large amount of memory space. According to the roofline model, the memory space occupancy is the sum of the parameter and the data size. Figure 5 shows the memory footprint of different offloading positions in the ResNET-34 Model [28]. ResNET-34 is a 34-layer ResNet, which is short for residual networks, a classic neural network which utilizes skip connections or shortcuts to jump over some layers, and is used for many computer vision tasks. It can be seen that as the offloading position gradually moves backward, the memory footprint increases slowly. But in the final offloading position, it has an exponential growth. It is because the last layers of the neural network model are usually fully connected layers, which occupies a lot of memory footprint.

In the roofline model, the theoretical computational performance that can be achieved on the computational platform of the edge nodes is closely related to the amount of computation and the amount of memory. Let the calculation amount of each layer of network model execution be Z, denoted as $Z = \{Z_1, Z_2 \ldots Z_n\}$. Let the memory space occupied by each layer of the network be D, denoted as $D = \{D_1, D_2 \ldots D_n\}$. First, we consider the calculation amount of the network model. For the sake of simplification, we ignore the bias parameter. The calculation amount Z_i (unit is FLOPS) performed by the ith layer network is calculated by the output feature map area Mi^2, convolution kernel area Ck_i^2, the number of input channels C_{i-1} and output C_i completely, as $Z_i = Mi^2 \cdot Ck_i^2 \cdot C_{i-1} \cdot C_i$. According to the calculation amount Z of each layer network, the total calculation amount Zt^j of the pre-j layer neural network can be obtained, denoted as

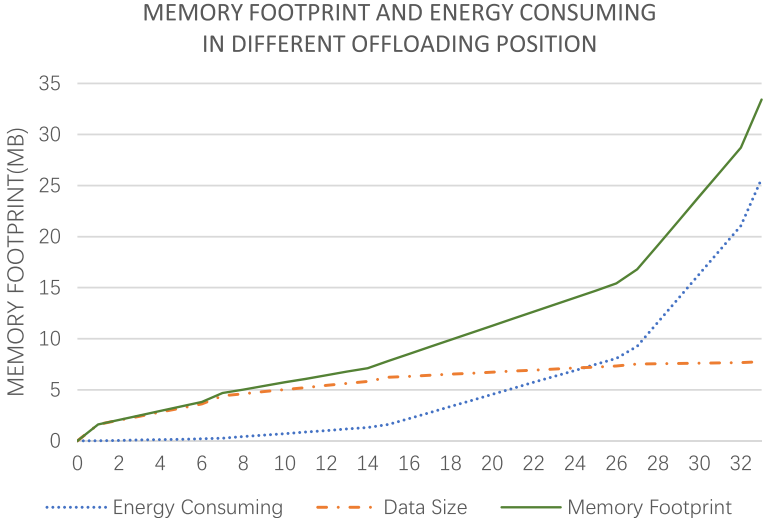

Fig. 5 Memory footprint of the model. The memory footprint of different offloading positions in the ResNET-34 Model

$$Zt^j = \sum_{i=1}^{j} Mi^2 \cdot \mathrm{Ck}_i^2 \cdot C_{i-1} \cdot C_i \tag{8}$$

where the output feature map area M_i^2 itself is determined by the input matrix size Ms_i, the convolution kernel size Ck_i, the pooling size Po_i, and the step size St_i, expressed as follows:

$$\mathrm{M}_i^2 = \left(\frac{\mathrm{Ms}_i - \mathrm{Ck}_i + 2 \cdot Po_i}{\mathrm{St}_i} + 1\right)^2 \tag{9}$$

For the memory footprint of the network model, it mainly includes two parts: the total parameter quantity and the output characteristic map of each layer. The parameter quantity is the total weight parameter of each layer of the model, and the feature map is the size of the feature image output by each layer of the model during the running process. The total parameter quantity Par_i of the ith network is related to the convolution kernel area Ck_i^2, the number of input channels C_{i-1} and the number of output channels C_i, as $\mathrm{Par}_i = \mathrm{Ck}_i^2 \cdot C_{i-1} \cdot C_i$. And the feature map size Map_i is only related to the output feature map area M_i^2 and the output channel number Ci, as $\mathrm{Map}_i = \mathrm{M}_i^2 \cdot Ci$. Calculating the memory footprint of the former j-layer neural network as Dt^j, then we know

$$Dt^j = \sum_{i=1}^{j} \mathrm{Par}_i + \sum_{i=1}^{j} \mathrm{Map}_i \tag{10}$$

If the memory footprint Dt_j of the former j-layer network is less than the memory space P of the edge node, the constraints can be met. It can be proved that the memory space of the edge pool depends on the minimum of edge nodes, so we assume the minimum values of the parameter $P_0 = \min\{Pv_1, Pv_2, \ldots, Pv_x\}$. Because the operating system in the edge node takes up a certain amount of memory space, we have previously defined some memory margin of the edge nodes. In this paper, we set a threshold that the memory footprint Dt^j of the former j-layer network is equal to λ_0=80% of the minimum memory space P_0, that is, the edge node memory is already saturated.

3.2 Task latency constraint

About the constraint on task latency, the maximum delay T_{max} allowed by the task is mainly composed of the edge side delay T_{es}, the cloud processing delay T_{cp}, the edge-to-cloud transmission delay T_{ec} and the terminal-to-edge uplink transmission delay T_{te}.

In our past work [7], we established the task latency model. In the edge side, since the distance between the edge nodes is very close, the communication delay is negligible. So, the edge side delay T_{es} only includes the task waiting and execution delay. We assume that there are K learning tasks that arrive in a certain period, denoted as $\{l_1, l_2, \ldots, l_k\}$. From the FLOP F_i of edge node v_j defined in II-B and the offloaded models' calculation amount Zt^j of task l_k, we can obtain the execution time as

$$t_{qp,e} = \int \frac{Zt^j}{F_i} dl_k \tag{11}$$

We assume that the task arrives independently; thus, the $M/M/N$ queuing model can simulate the queuing in the edge node. The waiting time can be calculated as

$$t_{qp,w} = \frac{a_k Zt^j}{F_i\left(F_i - a_k Zt^j\right)} \tag{12}$$

where a_k is the arrival rate of task l_k. The total task queuing and processing delay can be denoted as

$$T_{qp} = t_{qp,e} + t_{qp,w} \tag{13}$$

We assume that the cloud calculation rate is F_c, the cloud processing delay can be denoted as

$$T_{cp} = \int \frac{\left(Zt^n - Zt^j\right)}{F_c} dl_k \tag{14}$$

For the edge-to-cloud transmission delay T_{ec}, we can constrain it by the amount of task data uploaded. In a link with insufficient bandwidth, if the edge-to-cloud transmits a large amount of data, the task delay will be seriously affected. So, the amount of data transmitted by the task should also be constrained which is up to the communication network bandwidth W_H, and the throughput of edge nodes L. The maximum data output from the edge side to the cloud is:

$$C_0 = \min\{W_H, L_i\} \cdot T_{ec} \tag{15}$$

Since the edge-to-cloud data transmission amount is equal to the feature map size of the j th layer of the network Map_j, the amount of data is certain. We can get the edge-to-cloud data transmission delay T_{ec}.

For the terminal-to-edge uplink transmission delay T_{te}, each task can select the nearest edge node to process the task data, denoted as De. We have calculated the uplink delay of task l_k, as given by

$$T_{te} = \sum_{i=1}^{x} De_k \int \frac{p(vs_k)}{vs_k} dvs_k \tag{16}$$

where $p(F_i)$ denotes the probability density function (PDF) of variable vs_k, which can be calculated by a kernel density estimation method. If there is transmission rate as $Vs = \{vs_1, vs_2 \ldots vs_l\}$, then the PDF can be calculated as

$$p(vs_k) = \frac{1}{\lambda_k l} \sum_{i=1}^{l} \exp\left(- \left(\frac{vs - vs_{k,i}}{\lambda_k} \right)^2 \right) \tag{17}$$

where λ_k is the bandwidth parameter of the kernel used, as given by $\lambda_k = \sigma_k \left(\frac{4}{3l}\right)^{0.2}$. And σ_k is the estimated standard deviation of v_k.

Figure 6 shows the delay at the different offloading positions in the ResNET-34 Model. As we can see, the total delay decreases gradually as the offloading position moves backward.

3.3 Energy consumption and Lagrange multiplier optimization

Regarding the last constraint, because of the resource-constrained nature of edge computing, the energy of each node is usually limited. The energy consumption of the edge node includes static power consumption and computational energy consumption of the

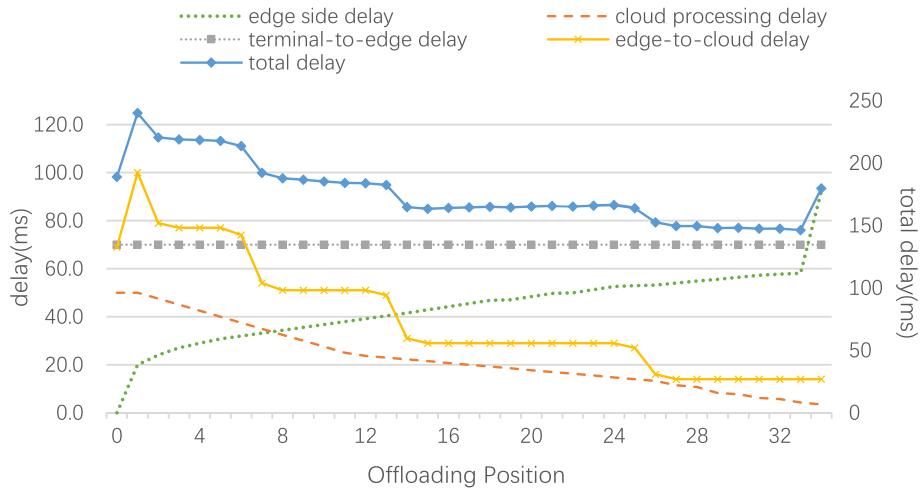

Fig. 6 Delay at the different offloading position. The time latency at the different offloading positions in the ResNET-34 Model

computational task. The static power consumption can be calculated according to the computational time of the task, denoted as

$$E_{i,k}^{s} = e_{i,k} \int \frac{Zt_{i,k}^{j}}{F_i} dl_{\mathrm{k}} \tag{18}$$

where $e_{i,\,k}$ denotes the unit time energy consumption of the node v_j in the task l_k, $Zt_{i,k}^{j}$ denotes the calculation amount of the node v_j in the task l_k, and F_i denotes the calculation speed of the node v_j. And the computational energy consumption can be calculated by

$$E_{i,k}^{c} = \omega_{i,k} Zt^{j} F_i \tag{19}$$

where $\omega_{i,\,k}$ denotes the energy consumption of unit calculation amount and speed. The energy consumption can be calculated by

$$E_i = \sum_{n=1}^{k} \left(E_{i,k}^{s} + E_{i,k}^{c} \right) \tag{20}$$

It will meet the constraint if the task energy consumption is less than the maximum energy consumption E allowed by the node. In summary, the j th layer is the optimal model offloading position, which needs to meet the following constraints:

$$P1: \ \arg\, max\, f(j) \ (0 \le j \le N) \tag{21}$$

$$\begin{aligned} s.t. \quad & C1: g(j) = \left(\sum_{i=1}^{j} Par_i + \sum_{i=1}^{j} Map_i \right) - \lambda_0 P_0 \le 0 \\ & C2: h(j) = \sum_{i=1}^{k} \left(T_{qp} + T_{ec} + T_{te} \right) - T_{\max} \le 0 \\ & C3: y(j) = \sum_{i=1}^{x} \sum_{k=1}^{n} \left(E_{i,k}^{s} + E_{i,k}^{c} \right) - \{E1, \ldots, Ex\} \le 0. \end{aligned} \tag{22}$$

That can be converted into Lagrange multiplier optimization:

$$L(j,\alpha,\beta,\gamma) = f(j) + \sum_{i=1}^{j}\alpha_i g_i(j) + \sum_{i=1}^{k}\beta_i h_i(j) + \sum_{i=1}^{x}\sum_{k=1}^{n}\gamma_{i,k} y_{i,k}(j) \quad (23)$$

With the Karush-Kuhn-Tucker (*KKT*) conditions of satisfaction, it can obtain a feasible solution by maximizing this function. The maximum value j is the optimal offloading position. Algorithm 2 summarizes the specific process of the task model offloading algorithm.

Algorithm 2 Task Model Offloading Algorithm

Function Offloading ($\{Ms_i, Ck_i, Po_i, St_i, C_i\}$, $\{a_k, \sigma_k, L, W_H, Dt, Vs,\}, \{e_{i,k}, \alpha_{i,k}\}, x, k, n, \lambda_0, F, P, E)$
$i \leftarrow 1$; $j \leftarrow 1$; $Zt^j \leftarrow 0$; $Dt^j \leftarrow 0$
while $i <$ N do
    $Calculate\ M_i^2$, Par_i , Map_i , Z_i , D_i
    $i \leftarrow i + 1$
while $j \leq$ N do
    $Dt^j \leftarrow Dt^{j-1} + D_j$
    $Zt^j \leftarrow Zt^{j-1} + Z_j$
    $Calculate \quad T_{qp}, T_{ec}, T_{te}, E_{i,k}^s, E_{i,k}^c, T_c$
    if$(Dt^j < \lambda_0 P_0) \,\&\, (T_{qp} + T_{ec} + T_{te} < T_{max})$ $\&\, (\sum_{n=1}^{k}(E_{i,k}^s + E_{i,k}^c) < E)$
        then $j \leftarrow j + 1$
        else break
    return $\{J \mid J := j - 1 (0 \leq j \leq N)\}$
End Offloading

4 Simulation results and discussion

In this section, we conduct an experiment to evaluate the effect of the collaborative cloud-edge computing framework. We assume that the actual task is about the camera sensor identifying the object class. To simulate the complex environment in the actual Internet of Things, the edge nodes select three different sets from {10,20,30} for testing, and we set a master node. We perform experiments using the tagged CIFAR-10 dataset [29], which consist of ten categories with 50,000 training images and 10,000 test images. Each image is a 224 × 224 color image, and each pixel includes three values of RGB, which is equivalent to three channels, and the value ranges from 0 to 255. In our experiment, we implement this framework in Python, with ResNET-34 network deployed in our framework. We experience the simulation tests on a laptop with python 3.6, 8GB RAM, Intel i5 1.6GHZ CPU, and Windows 10 operating system. Table 1 shows the offloading position of {10,20,30} nodes of ResNET-34 calculated by the task model offloading algorithm (TMOA). As can be seen, the TMOA algorithm can

Table 1 Offloading position calculated by TMOA

Number of nodes	Offloading position	Output layer
30 nodes	7th layer	3 × 3 conv, 064
20 nodes	13th layer	3 × 3 conv, 128
10 nodes	20th layer	3 × 3 conv, 256

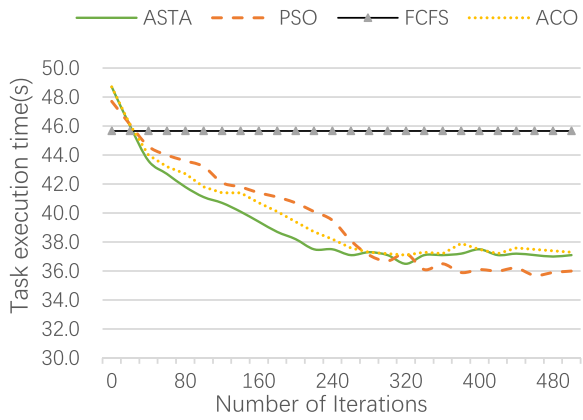

Fig. 7 Task execution time of FCFS, ACO and ATSA. The comparison of task execution time of the ATSA algorithm, the FCFS algorithm, and the basic ant colony optimization (ACO) algorithm in 10 edge nodes

determine the optimal offloading position, and with more edge nodes participate in the calculation, offloading position can be farther back.

As for the adaptive task scheduling algorithm (ATSA), we set pheromone weight $\alpha = 1.0$, heuristic information weight $\beta = 5.0$, pheromone volatility coefficient $\rho = 0.5$, and pheromone increments constant $Q = 5.0$. As shown in Fig. 7, we compare the task execution time of our ATSA algorithm with the FCFS algorithm [24] and the basic ant colony optimization (ACO) algorithm [25] and particle swarm optimization (PSO) algorithm in 200 tasks. The experiment shows that our ATSA algorithm has remarkably reduced the scheduling time than FCFS algorithm and could achieve faster convergence than other optimal algorithms like PSO algorithm.

Figure 8 compares the average data volume uploaded from only cloud computing scheme, collaborative cloud-edge computing scheme, and the scheme with cloud dormancy mechanism. It shows that our scheme can significantly reduce the amount of

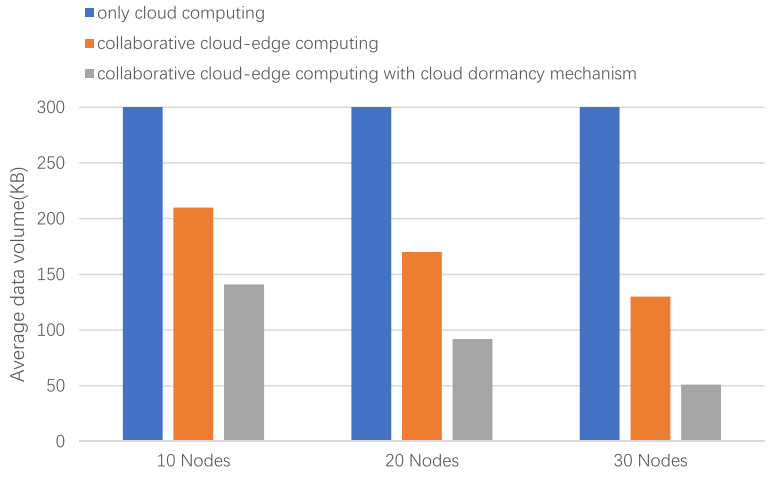

Fig. 8 Average data volume of different schemes. The comparison of the average data volume uploaded from only cloud computing scheme, collaborative cloud-edge computing scheme, and the scheme with cloud dormancy mechanism

Table 2 Delay and energy consumption

Model	Delay (ms)	Energy consumption(J)
ACEFLF (30 nodes)	25.7	42.5
ACEFLF (20 nodes)	40.2	46.7
ACEFLF (10 nodes)	57.1	49.6
B-Net (10 nodes)	69.1	62.9
MASM (10 nodes)	60.7	58.0

data uploaded to the cloud, that is, to reduce bandwidth usage and transmission delay, especially when the cloud dormancy mechanism is set.

Next, we compare the task delay and energy consumption of the framework of {10, 20,30} nodes with the BranchyNet model [20] of 10 nodes, MASM model [7] of 10 models, as shown in Table 2. We can see that with the number of edge nodes increases, the average delay and the energy consumption decline gradually. It is because the added nodes share the amount of computation. Because of the setting of cloud dormancy mechanism, the amount of data uploaded to the cloud is significantly reduced. Through the comparison of different frameworks in Table 2, we find that our framework performs better in delay and energy consumption than other existing frameworks of distributed neural network tasks.

Finally, we compare the accuracy of the framework with BranchyNet and the ResNet-34 [17]. For our framework, we set a weight decay of 0.0001, the momentum of 0.9. It starts with a learning rate of 0.1, which is divided by ten at 32k and 48k iterations. For each training, we randomly select 128 image data for small-batch training, and the total iterations are 64 thousand times.

Figure 9 shows the change in error rate as the number of iterations increases. It can be seen that our framework achieves the same effect as ResNet after about 30 iterations, which is better than BranchyNet.

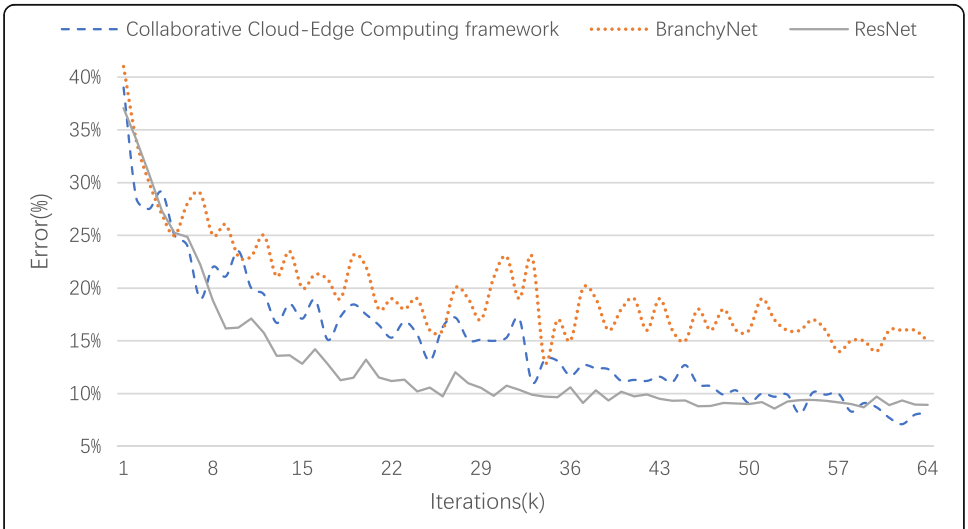

Fig. 9 Task accuracy of different frameworks. The error rate change of collaborative cloud-edge computing, BranchyNet, and ResNet framework as the number of iterations increases

5 Conclusion

In this paper, we have proposed a collaborative cloud-edge computing framework in distributed neural network, which focus on the neural network tasks in the resource-constrained IoT environment. We optimize the offloading position of task model by proposing a task model offloading algorithm (TMOA). We design an adaptive task scheduling algorithm (ATSA) to replace the FCFS mechanism for load-balancing of the edge nodes. We also propose a collaborative cloud-edge learning scheme, including the parameter aggregation scheme and the cloud dormancy mechanism. Experiments show that the framework achieves better results than existing other edge frameworks for the neural network task. A future direction is to develop a more efficient collaborative computing scheme that can be better deployed on the edge nodes.

Abbreviations

IoT: Internet of Things; DNN: Deep neural network; AI: Artificial intelligence; CNN: Convolutional neural networks; TMOA: Task model offloading algorithm; ATSA: Adaptive task scheduling algorithm; LSTM: Long-short-term-memory; MASM: Multiple algorithm service model; FCFS: First-come-first-served service; FLOPS: Floating-point operations per second; ACO: Ant colony optimization; KKT: Karush-Kuhn-Tucker

Acknowledgements

Not applicable.

Authors' contributions

SX and ZZ conceived and designed the study. SX and MK performed the simulation experiments. SX and MC wrote the paper. ZZ and MC reviewed and edited the manuscript. All authors read and approved the final manuscript.

Authors' information

Shihao Xu received the bachelor's degree in the School of Electronic and Information Engineering, Beijing Jiaotong University, in 2018. He is currently pursuing the Master's degree in Communication Engineering at the same university. His research interests include edge computing, data mining, and distributed deep learning.

Funding

This work is supported by The National Key R&D Program of China (2018YFC0831900).

Availability of data and materials

The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]School of Electronic and Information Engineering, Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University, Beijing, China. [2]School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China. [3]École de Technologie Supérieure (ÉTS), Montreal, QC H3C 1 K3, Canada.

Received: 18 October 2019 Accepted: 6 September 2020
Published online: 26 October 2020

References

1. A. Botta, W.D. Donato, V. Persico, A. Pescapé, Integration of cloud computing and Internet of Things: a survey. Futur. Gener. Comput. Syst. (2016)
2. F. Tao, Y. Cheng, L.D. Xu, L. Zhang, B.H. Li, CCIoT-CMfg: Cloud computing and Internet of Things-based cloud manufacturing service system. IEEE Transac. Indust. Inform (2014)
3. J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A survey on Internet of Things: architecture, enabling technologies, security and privacy, and applications. IEEE Internet Things J. (2017)
4. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges. IEEE Internet Things J. (2016)
5. X. Sun, N. Ansari, EdgeIoT: mobile edge computing for the Internet of Things. IEEE Commun. Mag. (2016)
6. T. Tuor, S. Wang, K.K. Leung and K. Chan, Distributed machine learning in coalition environments: overview of techniques. 21st International Conference on Information Fusion (FUSION), 2018
7. W. Zhang, Z. Zhang, S. Zeadally, H.-C. Chao, V.C.M. Leung, MASM: a multiple-algorithm service model for energy-delay optimization in edge artificial intelligence. IEEE Transac. Indust. Inform (2019)
8. S. Wang, T. Tuor, T. Salonidis, K.K. Leung, C. Makaya, T. He, K. Chan, Adaptive federated learning in resource constrained edge computing systems. IEEE J Sel. Areas Comm. (2019)
9. Y. Li et al., A 34-FPS 698-GOP/s/W binarized deep neural network-based natural scene text interpretation accelerator for mobile edge computing. IEEE Trans. Ind. Electron. (2019)

10. P. Paymard et al., Resource allocation in PD-NOMA–based mobile edge computing system: multiuser and multitask priority. Trans. Emerg. Telecommun. Technol. (2019)
11. Y. Chang, Research on de-motion blur image processing based on deep learning. J. Vis. Commun. Image Represent. (2019)
12. M. Gochoo, T. Tan, S. Liu, F. Jean, F.S. Alnajjar, S. Huang, Unobtrusive activity recognition of elderly people living alone using anonymous binary sensors and DCNN. IEEE J Biomed. Health Inform., 2019
13. H. Chen, P. Aggarwal, T.M. Taha and V.P. Chodavarapu, Improving inertial sensor by reducing errors using deep learning methodology. NAECON 2018 - IEEE National Aerospace and Electronics Conference. 2018
14. R. Girshick, Fast R-CNN. IEEE Int. Conf. Comp. Vision (2015)
15. S. Teerapittayanon, B. McDanel, H.T. Kung, Distributed deep neural networks over the cloud, the edge and end devices. IEEE Int. Conf. Distributed Comp. Syst (2017)
16. J.H. Ko, T. Na, M. F. Amir and S. Mukhopadhyay, Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained Internet-of-Things platforms, 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 2018.
17. K. Zhu, Z. Chen, Y. Peng, L. Zhang, Mobile edge assisted literal multi-dimensional anomaly detection of in-vehicle network using LSTM. IEEE Trans. Veh. Technol. (2019)
18. Z. Zhao, K.M. Barijough, A. Gerstlauer, DeepThings: distributed adaptive deep learning inference on resource-constrained IoT edge clusters. IEEE Trans. Comp. Aided Design Integ. Circuits Syst (2018)
19. Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, Neurosurgeon: collaborative intelligence between the cloud and mobile edge. 22nd ACM International Conference on Architectural Support for Programming Langua-ges and Operating Systems (ASPLOS), 2017
20. H. Xiao, Z. Zhang, Z. Zhou, GWS—a collaborative load-balancing algorithm for Internet-of-Things. SENSORS (2018)
21. N. Fernando, S.W. Loke, W. Rahayu, Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds. IEEE Trans. Cloud Computing (2019)
22. Y. Huang et al, Task scheduling with optimized transmission time in collaborative cloud-edge learning. 27th International Conference on Computer Communication and Networks (ICCCN), 2018.
23. P. Paymard et al., Joint task scheduling and uplink/downlink radio resource allocation in PD-NOMA based mobile edge computing networks. Phys. Comm. (2019)
24. W. Li and H. Shi, Dynamic load balancing algorithm based on FCFS. Fourth International Conference on Innovative Computing, Informa-tion and Control (ICICIC), 2009.
25. R. Xian-Jia, Research on hybrid task scheduling algorithm simulation of ant colony algorithm and simulated annealing algorithm in virtual environment. 10th International Confe-rence on Computer Science & Education (ICCSE), 2015.
26. S. Teerapittayanon, B. McDanel, and H. Kung. Branchynet: fast inference via early exiting from deep neural networks. 23rd Interna-tional Conference on Pattern Recognition, 2016.
27. G. Ofenbeck, R. Steinmann, V. Caparros, D.G. Spampinato and M. Püschel, Applying the roofline model. 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014.
28. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
29. A. Krizhevsky, *Learning multiple layers of features from tiny images* (2009)

Publisher's Note