Published in final edited form as:

*IEEE Internet Things J.* 2020 May ; 7(5): . doi:10.1109/jiot.2019.2963635.

# Towards Edge-Based Deep Learning in Industrial Internet of Things

**Fan Liang**[*], **Wei Yu**[*], **Xing Liu**[*], **David Griffith**[†], **Nada Golmie**[†]

[*]Towson University, USA

[†]National Institute of Standards and Technology (NIST), USA

## Abstract

As a typical application of the Internet of Things (IoT), the Industrial Internet of Things (IIoT) connects all the related IoT sensing and actuating devices ubiquitously so that the monitoring and control of numerous industrial systems can be realized. Deep learning, as one viable way to carry out big data-driven modeling and analysis, could be integrated in IIoT systems to aid the automation and intelligence of IIoT systems. As deep learning requires large computation power, it is commonly deployed in cloud servers. Thus, the data collected by IoT devices must be transmitted to the cloud for training process, contributing to network congestion and affecting the IoT network performance as well as the supported applications. To address this issue, in this paper we leverage fog/edge computing paradigm and propose an edge computing-based deep learning model, which utilizes edge computing to migrate the deep learning process from cloud servers to edge nodes, reducing data transmission demands in the IIoT network and mitigating network congestion. Since edge nodes have limited computation ability compared to servers, we design a mechanism to optimize the deep learning model so that its requirements for computational power can be reduced. To evaluate our proposed solution, we design a testbed implemented in the Google cloud and deploy the proposed Convolutional Neural Network (CNN) model, utilizing a real-world IIoT dataset to evaluate our approach[1]. Our experimental results confirm the effectiveness of our approach, which can not only reduce the network traffic overhead for IIoT, but also maintain the classification accuracy in comparison with several baseline schemes.

### Keywords

Industrial IoT; Edge Computing; Fog Computing; Distributed deep learning

## I. Introduction

The fourth industrial revolution, known as Industrial Internet of Things (IIoT), is a realization of the Internet of Things (IoT) [1], [2] in a variety of manufacturing systems, introducing a massive number of IoT devices and computation nodes in production lines

[1]Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

{fliang1,xliu10}@students.towson.edu .

and manufacturing processes so that the monitoring and control of manufacturing systems can be realized. In an IIoT system, as a typical cyber physical system, the key to realizing automation and intelligence is through big data analysis driven by big computing and big modeling provided by learning techniques such as deep learning [3]. In a traditional way, IoT devices collect data and send it to servers that have high computing capabilities for performing deep learning process. Then, the servers analyze the data and send control messages to IoT devices [4]. Due to the interactions between servers and IoT devices, massive amounts of data need to be transmitted through the IoT network, raising significant data transmission overhead to the network. As a number of IIoT systems are time sensitive, the large increase in network traffic causes high network latency and large packet loss, significantly affecting the performance of IIoT systems. Thus, how to optimize network performance while providing sufficient big data analytics becomes a critical problem in IIoT systems.

Edge (or Fog) computing has emerged as a new paradigm to offload computation tasks from the cloud to the edge. Unlike traditional cloud computing, in which tasks are offloaded to remote cloud datacenters, edge computing assigns computation tasks to multiple edge nodes that are deployed close to end users. Thus, edge computing is capable of reducing data transmission and network traffic between cloud servers and IoT devices (sensors, actuators, etc.) [5], [6], [7]. In an IIoT system, we can utilize edge computing to offload the computation tasks and reduce the network traffic as well. Although there are some existing studies toward increasing the network bandwidth or optimizing the data transmission, there is lack of research on how to carry out computing task offloading in IIoT.

Deep learning, as the useful big data-driven analytics scheme, has shown great potential in a number of areas, including image/video recognition, robotics, and natural language analysis, among others [8], [9]. Nonetheless, as deep learning requires high computation power to analyze the datasets, it is generally deployed in cloud servers, which have high computation capabilities. In addition, to obtain accurate results, large datasets are generally required. Thus, when deploying deep learning in the cloud to support IIoT systems, the massive data exchanged between servers and IoT devices could cause network congestion and affect IIoT systems that are commonly latency-sensitive. While deploying deep learning to the edge is a natural solution, it raises new challenges due to the limited computing ability of edge nodes. Thus, it is critical to design an effective deep learning model that can be used on edge nodes.

To address the aforementioned issues, in this paper we propose to leverage edge computing to conduct deep learning on edge nodes in IIoT systems. To demonstrate our idea, we design an IIoT scenario that utilizes the deep learning technique to classify different industrial components. We focus on offloading the deep learning process from the cloud to the edge so that network traffic congestion can be mitigated in IIoT systems. We design the edge-based Convolutional Neural Network (CNN) by leveraging the CNN model to classify the components. We propose a distributed CNN model where we deploy to edge nodes. The edge nodes execute the CNN training process and send the training results to the parameter server in the cloud. Using our designed model, we can significantly reduce the amount of the data transmitted through the network, leading to the improvement of network performance.

In our study, we make the following contributions:

First, we propose a novel edge-based CNN model to offload computation tasks. By doing this, IoT devices do not need to send raw data to a centralized server, thereby significantly reducing network traffic. Furthermore, in order to deploy the CNN model to the edge, we optimize the existing CNN model in Section IV. We also mathematically evaluate the time complexity of the proposed model and develop a mathematical model to illustrate how to deploy the proposed CNN model to edge computing nodes in Section IV.

Second, we evaluate network performance in an IIoT system. We design a mathematical model to analyze the network delay and packet loss rate in both edge-based CNN and centralized CNN cases. Overall, based on the specific IIoT scenario, we propose an edge-based CNN model to improve system performance. Since finding the optimal deep neural network configuration for a particular data set mathematically remains an open question, we leverage a combination of mathematical and experimental approaches to confirm the superiority of our model.

Third, we design an experimental testbed in Google Cloud to simulate the distributed environment in Section VI. To ensure a fair experimental comparison, we calculate the computation capability of edge nodes in the testbed. Thus, based on the calculation of time complexity for the different CNN models and the computation capability of the edge nodes, we can systematically analyze the performance of each CNN model and obtain meaningful results.

The remainder of this paper is organized as follows: In Section II, we conduct a brief literature review of related studies on IIoT systems and deep learning techniques. In Section III, we brief the key techniques of IIoT, edge computing, and deep learning. In Section IV, we introduce our approach in detail. In Section V, we define the scenario, introduce the testbed settings and experimental design, and define the evaluation metrics. In Section VI, we present the evaluation results. In Section VII, we discuss some further issues. Finally, we summarize the paper in Section VIII.

## II. Related Works

In the following, we review some existing research works that are relevant to our study. In the smart manufacturing system, the digital twin is a digital copy of real physical systems [10]. It is easy for operators and managers to emulate the operations on this digital system to avoid unexpected results. The key concept of the digital twin is utilizing massive data to create the digital model. For example, Qi *et al.* [11] reviewed massive data and digital twin, and compared differences between massive data and digital twin in manufacturing. Likewise, Canedo *et al.* [12] proposed a digital twin model to simulate the life-cycle of IIoT systems, which simulates IIoT services, objects, and the communications between the objects. Likewise, Tao *et al.* [13] proposed a framework of digital twin-driven product design and conducted a case study to evaluate its effectiveness.

Related to computing aspects of IIoT, fog/edge computing has been considered as a viable computing infrastructure to offload computation tasks in IoT [14]. For example, Peralte

*et al.* [15] proposed a fog computing-based scheme that introduces a low complexity computational layer between the cloud and IoT nodes. Yu *et al.* [5] conducted a comprehensive survey on edge computing and clarified how to leverage edge computing to support IoT. Li *et al.* [16] introduced the software defined network (SDN) to incorporate with edge computing and proposed an adaptive transmission architecture to improve network latency. Likewise, to optimize cloud computing in IIoT systems, Xu *et al.* [17] proposed a cloud-based architecture for IIoT systems and provided key services defined in different layers that are arranged in a cloud structure so that on-demand computing services with high reliability, scalability and availability can be supported.

Control of large-scale heterogeneous industrial systems remain a challenging problem so that powerful and efficient computation platforms and data analysis methods are necessary. As one of the most popular data-driven big modeling methods, deep learning techniques have been widely used in IoT and some existing studies have focused on utilizing deep learning techniques to assist in network control in IIoT systems. For example, Jiang *et al.* [18] utilized the deep learning techniques to improve the performance of the networks, such as massive Multiple Input Multiple Output (MIMO) antennas, ultra-dense small cell network, device-to-device communications, and so on. Likewise, Zhu *et al.* [19] utilized $Q$-learning to optimize the packet transmission schedule for IIoT applications. Furthermore, some research efforts aim to utilize the deep learning techniques to improve the performance of IoT applications. For instance, Mocanu *et al.* [20] designed different machine learning models to predict and classify the energy disaggregation task. Likewise, Huang *et al.* [21] investigated a deep learning-based scheme to perform forecasting of electrical loads.

In addition, some studies have been devoted to optimizing the performance of deep learning models, such as reducing time complexity and increasing accuracy, among others. For example, Zhang *et al.* [22] utilized the tensor-train deep computation model to compress hierarchical features so that more features can be trained in limited tensor space. Specifically, the tensor-train deep computation model compresses the features by converting the conventional dense weights to tensor-train format. By doing this, the proposed model could improve training efficiency and reduce memory space. Moreover, addressing the issue related to a lack of training samples, Zhang *et al.* [23] proposed an adaptive dropout to prevent deep learning models from overfitting, which is caused by a lack training samples. They designed a distribution function to determine the dropout rate of each layer. Then, a maximum entropy-based outsourcing selection algorithm was designed for selecting appropriate samples. Finally, they optimized the existing supervised learning model to fit the adopted adaptive dropout algorithm.

In contrast, two unsolved problems are tackled in our study. First, we focus on offloading the deep learning tasks from cloud servers to edge nodes, which reduces the amount of network traffic. Meanwhile, we optimize the deep learning model to reduce computation requirements and improve execution on edge nodes. Second, we utilize the distributed deep learning model to address the manufacturing components classification problem. We design an IIoT scenario and select the real-world dataset to validate the effectiveness of our model.

## III. Preliminaries

In this section, we introduce the topics of IIoT, edge computing, and deep learning.

### IIoT:

Generally speaking, IIoT provides the network infrastructure for connecting IoT devices so that the monitoring and control of industrial manufacturing systems can be supported. From a cyber-physical system perspective, it is composed of both the physical subsystem and the cyber subsystem, which interact with each other so that the manufacturing process can be monitored and controlled with the aid of advanced information communication techniques. By interacting with computing and networked objects in the physical subsystem, IoT devices (sensors, actuators, etc.) collect data, utilize the network subsystem to transmit the data to the operation center, in which the data will be further analyzed to assist system decision making, and receive data to conduct actuation and modification of physical assets. As a kind of distributed system [24], all IoT devices in IIoT systems connect via communication networks. In IIoT, as numerous applications are time-sensitive, network performance is the key factor that affects the performance of IIoT applications. Nonetheless, to support automation and intelligence for IIoT applications, a large amount of data will be collected and analyzed. While more data can provide better intelligence to IIoT applications, transmitting massive data through the network could lead to network congestion and further affect the monitoring and control performance of IIoT applications.

### Edge Computing:

Edge computing, with a similar scope to fog computing, which extends cloud computing to the network edge, is a distributed computing architecture to offload computation tasks from the cloud to edge nodes that are close to end-users [5], [6]. Moreover, edge computing offers latency reduction benefits for some time-sensitive applications. Thus, it is viable to leverage edge computing to support IIoT so that big data analysis tasks can be offloaded and the amount traffic transmission can be reduced. Nonetheless, edge nodes have limited computation power and generally cannot handle highly extensive computation tasks. Furthermore, the communication and synchronization of edge nodes could affect the performance of edge computing, as computation tasks are distributed to heterogeneous edge nodes that must cooperate. Thus, how to reduce the computation demand and transmitted traffic overhead to the network are key issues for edge computing-based IIoT systems.

### Deep Learning:

As we discussed above, the key to automation and intelligence for IIoT is data analysis. One of the most popular data-driven modeling technique is deep learning. A number of deep learning techniques have been widely deployed in regression, classification, and forecasting [8], [25], and have shown greater potential compared with other data analysis schemes. Generally speaking, the deep learning model is fed a training dataset and utilizes various methods of gradient descent. Nonetheless, the complexity, diversity, and integrity of the training dataset could significantly affect training results. Training with sufficiently larger datasets could result in more accurate output from equivalent models

(shape, layers, and activation, among others). Nonetheless, training on large datasets requires high computational power.

Furthermore, the data is continuously collected and increasing, and the demands of computation increase accordingly. Thus, it is difficult to handle such a task with only one computation node. There are two possible ways to tackle this issue. One way is to optimize the deep learning model so that the computation requirement can be reduced. The other is to distribute the deep learning model to a group of computation nodes, in which distributed learning is conducted. As the computation can be subdivided and distributed, the computation time in total could be improved.

## IV. Our Approach

In this section, we introduce our approach in detail. Particularly, we first outline the design rationale, detail the system models, and compare the performance of cloud-based and edge-based deep learning schemes. We then propose our edge-based deep learning model. Table I lists key notations in this paper.

### A. Design Rationale

Based on the discussion above, we formalize the problem of IIoT systems. Fig. 1 illustrates the problem space of IIoT systems, which consists of three dimensions (i.e., network, computation, and system structure). The solid blue sectors in the figure indicate our area of focus. In this study, we focus on utilizing edge computing to offload deep learning from the cloud, as it can reduce network traffic and mitigate congestion. Furthermore, we optimize the deep learning model and reduce the computation requirements of the deep learning process to deal with edge nodes that have less computation power than cloud servers. Our goal is to design an optimized deep learning model that is tailored to the edge computing platform so that both computation time and network latency can be reduced.

We now introduce our design rationale which focuses on the following issues:

**Network Performance:** The network system in IIoT provides communication infrastructure for the data exchange between subsystems. Moreover, in the centralized IIoT system, both control and data analysis processes are maintained in the datacenter cloud, which is denoted as the cloud-based system. In such a system, the raw datasets need to be uploaded to the datacenter, resulting in large data flows that can occupy the network resources and affect control signal transmission. As the control signal is the core heartbeat in the IIoT system, variability in the transmission of control signals could significantly affect the entire IIoT system. Thus, the effectiveness of the network system directly affects the performance of the IIoT system as a whole and is a key factor in IIoT. To this end, we focus on the design of an edge computing-based system to offload the data analysis process from the cloud to the edge so that the amount of network traffic can be reduced.

**Deep Learning Model Performance:** Deep learning is a popular data-driven modeling scheme, which has shown great potential in IIoT systems. Powerful computation support is required for the training process of deep learning models, in order to obtain accurate results.

Commonly speaking, edge nodes have less computation power than the centralized cloud servers. If the data analysis process is moved from cloud to edge, we need to optimize the deep learning model so that the demands of computation power can be reduced. The complexity of deep learning model implementation determines the computation requirements in the training process. For instance, in the CNN model, the number of convolutional layers affect the complexity directly. Thus, we focus on reducing the number of convolutional layers and optimizing their size to reduce the complexity of the CNN model while maintaining equivalent performance.

**System Performance:** As we discussed above, deploying deep learning to the edge and reducing the computation requirements of the deep learning model are two viable directions to improve the performance of the IIoT system. As each approach could affect system performance, to improve the entire system performance, we need to realize both approaches simultaneously.

In the following, we first propose an optimized CNN model to reduce computation cost. We then deploy the proposed CNN model to the edge node and compare its performance with the cloud-based CNN model as a baseline for comparison.

## B. Deep Learning Model

**1) Dataset Selection:** In an IIoT environment, one important task is the detection and identification of different industrial objects from images and video, which may be produced in a variety of applications. To evaluate the performance of the proposed model, we choose the T-Less dataset [26], which is a set of images of different industrial components. The T-Less dataset is a public dataset of 6D posed texture-less rigid objects. The T-Less dataset includes over $10^5$ images with 30 different industrial components captured by cameras with fixed angles. Each component includes 1,260 '*.png*' image files at 480 × 480 pixels each. The training images contain each individual object with a black background, while the test images show twenty table-top scenes with arbitrarily arranged objects [26]. The images show different angles of each component with $RGB$ channels. Fig. 2 shows some samples from the dataset. From these examples, we can see that some components have similar shapes or sizes, such as objects 5, 6 and 10. In addition, some components are assembled by other components, such a0s object 9, which is assembled from objects 6 and 10. This is common in industrial environments, where some components are parts of others. This unique characteristic of the dataset can help us evaluate the applicability of our algorithms in real-world industrial environments.

**2) CNN Model Design:** We now introduce the CNN model design. First, we identify the deep learning model. As illustrated above, the training datasets are images in our case, and utilizing the CNN model is one of the general approaches for image processing [27]. Thus, based on the conventional CNN model, we propose our optimized model in our IIoT scenario. Since the training process of the CNN model is a black box process [28], we need to define and optimize the parameters of the model. The number of convolutional layers is an important parameter in a CNN model, affecting performance directly [29]. To find an effective neural network structure, we create several models with different numbers

of convolutional layers and compare the performance. Also, the learning rate is another important parameter for the CNN model that we need to tune, as it affects the convergence speed of the model. Identifying a suitable learning rate for a CNN model could enable faster convergence and desirable accuracy. In the following, we discuss the details of parameter optimization.

**Number of Convolutional Layers:** Figs. 3 and 4 illustrate the performance of the CNN models with different numbers of layers. Particularly, Fig. 3 shows the classification accuracy of the CNN models with different numbers of convolutional layers. Here, the *x*-axis represents the number of convolutional layers and the *y*-axis represents the classification accuracy. Note that the popular CNN model, LeNet-5 [30], has 6 convolutional layers and 2 full-connection layers, and another popular CNN model, VGG-16 [31], has 16 convolutional layers and 1 full-connection layer. It is also worth noting that existing studies have shown that these CNN models perform better than most on image processing tasks [32], [33]. Thus, we use VGG-16 as a baseline model, which is known as one of the most accurate CNN model [31]. The experimental results of our comparison illustrate that classification accuracy increases rapidly when the number of convolutional layers increases from ‘1’ to ‘4’. When the number of convolutional layers is larger than ‘4’, the classification accuracy observes no significant change. Fig. 4 illustrates the relationship between the training speed and the number of convolutional layers. We find that the training speed continually drops as the number of convolutional layers increases, but the rate of decrease drops as well. Thus, based on the performance, we identify the number of convolutional layers for our case is ‘4’.

**Learning Rate:** The learning rate is another important parameter in the CNN model that controls how much the model can adjust the weights of the neural network with respect to the loss gradient. Table II illustrates the identification process for the learning rate. Setting the learning rate to ‘0.05’ and ‘0.01’ results in the gradient divergence, making the loss of the model approach infinity (NaN). In the opposite, setting the learning rate between ‘0.005’ to ‘0.001’, the convergence speed decreases. From the table, we observe that, while setting the learning rate to ‘0.005’ obtains the faster convergence speed, the convergence process is not stable. Thus, we set the learning rate to ‘0.004’ for our model.

### 3) CNN Model Analysis:

In the following, we analyze the proposed CNN model in detail.

**Time Complexity:** We now analyze the time complexity of the proposed CNN model. Fig. 5 illustrates the structure of the CNN model. As we mentioned before, we select four convolutional layers to extract feature maps. Moreover, we deploy a pooling layer in the model following each convolutional layer (i.e., the total number of pooling layers is four), in order to further compress the feature maps. Based on the mathematical definition of both the convolutional and pooling layers, the total computation cost of the CNN model is

$$T = \sum_{i=1}^{n} \left[ c \cdot \left( L_{K_i}^2 + 1 \right) \cdot L_{I_i}^2 + \left( L_{I_{i+1}} / L_{P_i} \right)^2 \right] \cdot m_i .$$

(1)

Here, we formalize the computation cost of the CNN model by utilizing the number of $RGB$ channels for the input images, the number of filters, and the number of the convolutional layers (all symbols in the equation have been defined in Table I). Meanwhile, we assume the length of the convolutional kernel is $L_{K_i}$ and the length of the input features is $L_{I_i}$, where $i$ represents the number of inputs. Also, we denote $L_{P_i}$ as the size of the pooling layer. From Equation (1), the time complexity of the CNN model can be represented by

$$O\left(\sum_{i=1}^{n} L_{K_i}^2 \cdot L_{I_i}^2 \cdot m_i\right).$$

(2)

By utilizing Equation (1), we are able to evaluate the number of computations required for each CNN model. The proposed CNN model has 4 convolutional layers and the number of calculations for the proposed CNN model is approximately 2 million per image. In addition, the LeNet-5 model has 6 convolutional layers and VGG-16 model has 16 convolutional layers. Based on the calculation, the number of computations for LeNet-5 and VGG-16 are approximately 7.5 million calculations per image and 31 million calculations per image, respectively. The results clearly show that the number of computations carried out by the proposed CNN model is the lowest, directly correlating to a reduction in computation overhead.

**Forward Propagation:** We analyze the forward propagation of the proposed CNN model. The CNN model consists of two components: one is the convolutional layer and the other is the fully connected layer. The forward propagation for the convolutional layer can be denoted as

$$\mathbb{C}_{n+1} = f\left(\sum_{s=0}^{m-n}\sum_{s=0}^{n} \mathbb{I}_{(n,n)}^{(s,s+n)} \odot \mathbb{K}_{(n,n)} + b\right),$$

(3)

$$P_{n+1} = \max(\boldsymbol{C}_{n+1}),$$

$$f(z) = softmax(z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_i}}.$$

Here, $\mathbb{C}_{n+1}$ denotes the results after the prior convolutional layer, $\mathbb{I}$ denotes the input and $m$ denotes the size of the input, $\mathbb{K}$ denotes the convolution kernel, and $n$ denotes the size of the convolution kernel. The expression $\mathbb{I}_{(n,n)}^{(s,s+n)} \odot \mathbb{K}_{(n,n)}$ represents convolution operation, and the operator " $\odot$" is the dot product for each element at the corresponding location in matrix $\mathbb{I}$ and $\mathbb{K}$. Also, $s$ indicates the start location of the convolution calculation and $f(z)$ is the softmax function, where $k$ is the length of $z$. We assume that the step size is '1' and $b$

indicates the bias. We utilize the maximum pooling function to further compression the feature size.

The forward propagation for the full connect layer can be represented by

$$z_{n+1} = w_{n+1}a_n + b_{n+1}, \tag{4}$$

$$a_{n+1} = f(z_{n+1}),$$

$$f(z) = ReLU(z) = \max\{0, z\}.$$

Here, we denote $n$ as the layer number, $z$ as input, $a$ as output, and $f(z)$ as activation function. We use the Rectified Linear Unit function (ReLU) as the activation function.

**Backward Propagation:** Backward propagation is one key feature for deep learning, as it updates the weights and bias in order to tune the model and obtain accurate results. The backward propagation is related to the partial derivative. For the proposed CNN model, the backward propagation of our CNN model can be represented by

$$\delta_{n-1} = \delta_n \frac{\partial z_n}{\partial z_{n-1}} = \delta_n \cdot rot180(\mathbb{K}_{n,n}) \odot f'(z_{n-1}). \tag{5}$$

Here, $rot180$ indicates the rotation of the convolution kernel by 180 degrees.

The backward propagation for the fully connected layer is the partial derivative for the softmax regression, which can be represented by

$$\frac{\partial a_j}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_j}}{\sum_{k=1}^{T} e^{z_k}}\right) = \begin{cases} a_j(1-a_j), & \text{if } j = i; \\ -a_j a_i, & \text{if } j \neq i. \end{cases} \tag{6}$$

Note that the above equation can be simplified in two cases, i.e., $j = 1$ and $j \neq i$.

**4) Edge-Based CNN Model:** Based on our proposed CNN model, we tailor the model to operate in a distributed manner and deploy the model in edge computing nodes. The new model is denoted as the edge-based CNN model. Generally speaking, the proposed edge-based CNN model is essentially a data parallel distributed CNN model that generates a CNN graph and assigns it to edge nodes. By doing this, all the edge nodes utilize the same CNN graph and are fed with different data subsets. Then, the training parameters are uploaded to the parameter server (i.e., aggregation node) to update the model. Thus, the

edge-based CNN model can offload the data analysis process from the cloud to the edge so that the network traffic in the IIoT network can be reduced.

In this study, we utilize synchronous stochastic gradient descent to update the 'weights' and 'bias' for all the workers as edge nodes. We set $n$ as the number of workers and $m$ as the number of samples trained on one worker. In the following, we show that the edge-based CNN model has the same convergence process as the cloud-based CNN model.

Here, we use the following equation to represent the mathematical calculation of the synchronous stochastic gradient descent, which executes the training on a single machine,

$$\mathbb{K}_{i+1} = \mathbb{K}_i - \frac{\alpha}{n \cdot m} \sum_{j=1}^{n \cdot m} \frac{\partial Loss_j}{\partial \mathbb{K}_i}. \tag{7}$$

Here, data size is $n \cdot m$ and the learning rate is $\alpha$ . Also, we distribute the model to $n$ nodes and assign dataset blocks of size $m$. Thus, we obtain

$$\mathbb{K}_{i+1} = \frac{1}{n} \sum_{w=1}^{n} \mathbb{K}_{i+1,w}, \tag{8}$$

$$= \frac{1}{n} \sum_{w=1}^{n} \left( \mathbb{K}_i - \frac{\alpha}{m} \sum_{j=(w-1)m+1}^{wm} \frac{\partial \mathrm{Loss}_j}{\partial \mathbb{K}_i} \right), \tag{9}$$

$$= \mathbb{K}_i - \frac{\alpha}{n \cdot m} \sum_{j=1}^{n \cdot m} \frac{\partial \mathrm{Loss}_j}{\partial \mathbb{K}_i}. \tag{10}$$

## C. Performance Analysis

We now analyze the performance of the cloud-based and edge-based models. In our case, we use the total processing time as the metric for system performance. The total processing time consists of two parts: (i) computation process time, and (ii) network transmission time. In the following, we will model and analyze these two parts individually.

**1) System Latency:** To measure network performance, we measure the system latency, which is defined as the total time of data transmission and deep learning processing for both models. Our analysis shows the edge-based models always has less delay than cloud-based model.

**Cloud-Based Model:** For the cloud-based model, the system latency can be represented by

$$T_d = T_c + T_{t1} + T_r + T_{t2} + T_p, \tag{11}$$

$$T_{t1} = \frac{D_s}{R_1},$$

$$T_{t2} = \frac{n \cdot D_s}{R_2},$$

$$T_p = f(mn \cdot D_s) + c_r.$$

We assume the collection time $T_c$ and receiving time $T_r$ are constant and can be represented by $T_c = c_c$ and $T_r = c_r$. We also denote $D_s$ as the amount of data that is collected by each sensor, and $R_1$ as the data rate of the upload link from the sensor to the edge node. Finally, we denote $m$ as the number of edge nodes and obtain the process time $T_p$, and further denote $f(x)$ as the time complexity of the deep learning algorithm. Recall that, $T_d$, $T_c$, $T_{t1}$ , $T_r$, $T_{t2}$, and $T_p$ have been defined in Table I.

**Edge-Based Model:** For the edge-based model, we offload the deep learning from the cloud to the edge. Thus, the edge nodes execute the deep learning algorithm instead of the cloud server. This can be formalized by

$$T_d^e = T_c + T_{t1} + T_p^e + T_{t2}^e + T_r^e, \tag{12}$$

$$T_p^e = c_r + (m+1) f(n \cdot D_s),$$

$$T_{t2}^e = \frac{D_e}{R_2}.$$

Similar to the cloud-based model that we have already defined, $T_d^e$ is the system delay. The differences are that $T_p^e$ represents the process time on edge nodes and $T_r^e$ represents the receiving time cost for the cloud server, which is a constant $c_r^e$. In the edge-based model, $T_p^e$ equals the sum of the receiving time $c_r$ and the training time. $D_e$ represents the output data size for each edge node, and $R_2$ is the data rate of the link from the edge node to the cloud. Thus, the transmission time $T_{t2}^e$ is the data amount divided by data rate.

**2) Network Overhead:** In our case, the data transmission time can be represented by

$$\Delta T_{net} = T_{t2}^{e} - T_{t2}\,. \tag{13}$$

We compare the network overhead of the two models by computing the difference $\Delta T_{net}$ between them. Because the proposed CNN model reduces the input image size as we discussed in Section IV-B3, $D_e$ is always smaller than $nD_s$ , and $\Delta T_{net} < 0$, indicating that the edge-based model achieves better network performance than the cloud-based model.

**3) Computation Overhead:** In the edge-based model, the computation time $T_{com}^{e}$ is

$$T_{com}^{e} = T_{p}^{e} + T_{r}^{e} = c_r + (m+1) f(n \cdot D_s) + c_r^{e}, \tag{14}$$

In the cloud-based model, the computation time $T_{com}$ is

$$T_{com} = T_r + T_p = c_r + f(mn \cdot D_s) + c_r\,. \tag{15}$$

According to Equation (2) in Section IV-B3, the time complexity of our deep learning algorithm $f(x)$ equals $\theta(n^2)$. Then, we compare the system latency of two models by computing the difference $\Delta T$ between these two models, which can be represented by

$$\Delta T = T_{com}^{e} - T_{com}, \tag{16}$$

$$= (m+1) f(n \cdot D_s) - f(mn \cdot D_s) + (c_r^{e} - c_r), \tag{17}$$

$$= (m+1) n^2 D_s^2 - m^2 n^2 D_s^2 + (c_r^{e} - c_r), \tag{18}$$

$$= -\left(m - \frac{1}{2}\right)^2 (nD_s)^2 + \frac{5}{4}(nD_s)^2 + (c_r^{e} - c_r), \tag{19}$$

$$= -(nD_s)^2 \left[\left(m - \frac{1}{2}\right)^2 - \frac{5}{4} - \frac{(c_r^{e} - c_r)}{(nD_s)^2}\right].$$

(20)

Thus, we have

$$\forall m \in \mathbb{N} \wedge m > \sqrt{\frac{c_r^e - c_r}{n^2 D_s^2} + \frac{5}{4}} + \frac{1}{2}:$$

(21)

$$(m+1)n^2 D_s^2 - m^2 n^2 D_s^2 - (c_r^e - c_r) < 0.$$

This indicates that when $m > \sqrt{\frac{c_r^e - c_r}{n^2 D^2} + \frac{5}{4}} + \frac{1}{2}$, the edge-based model always has better performance than the cloud-based model.

## V. Implementation and Experimental Design

In this section, we introduce the implementation and experiments to validate our approach in detail. In the following, we first define the scenario, and setup the testbed on Google Cloud Instances [34] based on the scenario. Then, we design a set of experiments to evaluate the performance of the proposed model. Finally, we describe several evaluation metrics of the experiments.

### A. Scenario

According to Section IV, we now define one representative scenario in IIoT. In a smart factory, ‘30’ different kinds of components are produced by different production lines and the system is required to classify the different components in the assembly center so that the product can be assembled properly. To do this, the cameras are deployed in different production lines, taking the images of components and sending the images to gateways. We denote those gateways as edge nodes. Then, edge nodes send all the images to the cloud for further analysis in order to prepare the classification. In this case, sending all the images to the cloud could result in significant network traffic congestion by consuming substantial network resources. Furthermore, uploading all the data to the cloud increases the total data processing time significantly because of the transmission time cost.

In our scenario, we focus on offloading the data analysis process from cloud to edge nodes (i.e., gateways). We involve a total of ‘3’ edge nodes and each node receives images from ‘10’ different production lines. Specifically, components ‘1’ through ‘10’ are processed by Edge Node ‘1’, components ‘11’ through ‘20’ are processed by Edge Node ‘2’, and components ‘21’ through ‘30’ are processed by Edge Node ‘3’. All edge nodes are connected by wired networks. After receiving the images from the production lines, the edge nodes begin with the data analysis process and upload only the analytical results to the cloud instead of the raw dataset. In our case, the training process of the CNN model is offloaded to the edge computing nodes and the edge nodes obtain the well-trained model

after training. Then, the nodes send the well-trained models to the cloud, and the cloud classifies the different components by using the well-trained models.

## B. Testbed Settings

To implement the scenario and proposed edge-based CNN model, a suitable and high-performance computing platform is important. Recall that the proposed model in Section IV-B requires an edge computing platform and high computation power for each computing node. Furthermore, to compare the training performance between cloud-based and edge-based deep learning, the computation power should be flexible and adjustable. Based on the requirements, we utilize Google Cloud Instances [34] and configure a group of instances to form a hierarchical computing network.

**Computing Node Configuration:** We utilize five Google Cloud Instances as the computing nodes in our testbed. Three are defined as edge nodes and the others are defined as cloud servers, which are shown in Fig. 6. Meanwhile, according to the different roles of the computing nodes in distributed deep learning, we define one of the cloud servers as the parameter server and the other as the master node. The three edge nodes are worker nodes. All the instances are configured with 8 core Intel CPU, 32GB memory and each instance is running Ubuntu 18.04 Long Term Support (LTS) operating system. We selected TensorFlow Application Programming Interface (API) and Python to realize the proposed model. We installed Anaconda 3, which is a popular Python distribution for deep learning and data science on the Linux platform. The TensorFlow virtual running environment has built using TensorFlow 1.12, Keras 2.2.4, and Python 3.6. Finally, we deployed the Python code to the different nodes.

**Network Configuration:** As we discussed in Section IV-B, the workers in the distributed CNN model need to connect with the parameter server and the master node in order to exchange weights $w$ and bias $b$. The IP address 10.132.0.1 is configured as the gateway. Then, we assign the computing nodes with IP addresses in the network segment to ensure all nodes are able to connect with the others. The instances in the Google cloud are organized by wired network, and we define the roles (parameter server and worker) of the different instances in the Python code and assign IP addresses accordingly.

**Computation Power Estimation:** Since the computation power is difficult to calculate and the performance depends on the hardware and software, to be fair, we utilize the proposed CNN model to estimate the computation power for several Google Cloud Instances with different hardware. In particular, we first set the instance with a 4 core CPU and 16GB memory at the beginning and execute the CNN model to obtain the running speed. After that, we update the CPU to 8 cores and to 32GB memory, and then increase the number of cores and memory size until reaching 32 cores and 128GB memory. Fig. 7 illustrates the results of computing speed for several Google Cloud Instances with various hardware configurations. The $x$-axis is the hardware configuration and the $y$-axis shows the number of training steps completed in one second. As we configure the edge-based testbed with three edge nodes (workers), to compare the performance between the edge-based architecture and cloud-based architecture, the total computation power should be the same for both

architectures. Based on our configuration for the edge-based testbed that has 8 CPU cores and 32GB memory, according to the estimation shown in Fig. 7, we select 32 core CPU and 128GB memory for the cloud-based testbed.

### C. Experimental Design

Following the testbed setup and configuration, we design experiments to evaluate our edge-based CNN architecture. In the following, we introduce the design of experiments in detail. We first focus on the defined IIoT scenario, which utilizes the industrial components dataset for training and testing. We then optimize the edge-based CNN model and evaluate its performance.

**1) Data Preparation:** Based on the T-Less dataset that we described in Section IV-B and the designed testbed, we first compress the images to 128 × 128 pixels, in order to reduce the computation pressure on the computing nodes. Furthermore, since all the images have black backgrounds and the objects are located in the center of the image with no texture and color, we transform all the images from three-channel $RGB$ to single-channel grayscale images. The preprocessing is able to significantly reduce the data size and the computation amount. Then, we randomly select 1000 images as the training dataset and 260 images as the testing dataset from each component. After identifying the training and testing datasets, we use the labels ‘1’ through ‘30’ to mark the different components. For the cloud-based model, we utilize all ‘30’ components as the training and testing datasets, while for the edge-based model, we divide the dataset into three parts: labels ‘1’ to ‘10’, labels ‘11’ to ‘20’, and labels ‘21’ to ‘30’. We then use the different training and testing datasets to train the different edge computing nodes. Finally, the standardization input dataset ‘tfrecord’ files are generated by the Python program.

**2) Deployed Models:** We now present the deployed models. First, we implement the CNN model on a single Google Cloud Instance as the cloud-based CNN model. We set the instance with ‘3’ CPUs, where each CPU has ‘8’ cores and the total system memory is 128GB, the same computation power as the three edge workers in the edge-based CNN model described in Section V-B. We evaluate the performance of the cloud-based CNN model, which we utilize as a baseline for comparison. Then, we deploy the edge-based CNN model on the testbed and compare its performance with the baseline model. Second, we modify several mature CNN models, such as LeNet-5 [30] and VGG-16 [35], to run in a distributed manner, and deploy them in the testbed. We then compare the performance between our proposed CNN model and the existing CNN models.

### D. Metrics

Based on the outlined scope and experimental design, we consider the following performance metrics to evaluate the proposed CNN model.

**Training Loss:** The training loss is one important metric for deep learning models. We select the softmax function to classify the image and the cross entropy loss function to evaluate the total loss. The softmax regression can be represented by $P_j = \frac{e^{a_j}}{\sum_{k=1}^{T} e^{a_k}}$, which

indicates the probability that the output belongs to the $j^{th}$ classification. Here, $T$ indicates the number of the classifications, $a_j$ is the $j^{th}$ element in the $T \times 1$ vector, and in our scenario, $T$ is set to 10. Then, we obtain the cross entropy loss function: $Loss = -\sum_{i=1}^{T} y_i$, In $a_i$ where $y_i$ represents the real value and $a_i$ represents the result from the softmax regression. In general, a smaller loss value indicates a better model.

**Classification Accuracy:** The classification accuracy is another important metric for evaluating deep learning models and represents the success rate of the classification. Higher classification accuracy means better performance of the model. In our evaluation, we select classification accuracy as the evaluation metric to measure the performance of the purposed CNN model.

**Time Cost:** Another important metric for the evaluation is the total time cost. Recall in our edge-based model, we define the total processing time $T_d^e = T_c + T_{t1} + T_p^e + T_{t2} + T_r^e$, and the time cost difference between cloud-based and edge-based models appears in both training time $T_p^{'}$ and transmission time $T_{t2}$. The training time represents the efficiency of the CNN model, while the transmission time represents the output size of the CNN model. Thus, both the training time and transmission time are key metrics to quantify the performance of the CNN model.

## VI. Evaluation Results

We now detail the evaluation results of the experiments outlined in Section V. In the following, we first present the comparison of cross entropy loss and classification accuracy between the cloud-based and edge-based CNN models. We then present a comparison of results obtained from the proposed edge-based CNN model and several existing CNN models. Because of the program initializes ‘weights’ and ‘bias’ randomly in the experimentation. Thus, the experiment results may be different. To be fair regarding the experimental results, we run the program 10 times and obtain experimental results to draw the error bar with 95% confidence intervals. Finally, we show the training time consumption for different models.

### A. Cloud-Based CNN Model vs. Edge-Based CNN Model

As we discussed in Section V-C, we deploy the proposed CNN model in both cloud-based and edge-based environments. For the fairness of comparison, the computation power of the two environments is the same. Fig. 8 illustrates the comparison of cross entropy loss between cloud-based and edge-based CNN models. In the experiments, we execute the edge-based CNN model 10 times and set the confidence interval. Because the initialization of the CNN model assigns weights and bias randomly in the Python code, the training results may differ upon each execution of the CNN model. The evaluation results show that the edge-based CNN model achieves larger loss values before 500 training steps, and then the loss of the two CNN models is approximately the same. The loss value in the $3000^{th}$ step is 0.00135 for the cloud-based model and 0.00151 for the edge-based model. Thus, the two models have equivalent performance, the only difference is that the edge-based model achieves convergence at a slower speed prior to 500 training steps.

Fig. 9 illustrates the classification accuracy for the two models, which we have divided into three parts: (i) training steps 0 to 500, (ii) training steps 500 to 2000, and (iii) training steps 2000 to 3000. In the first part, the cloud-based model achieves higher accuracy than the edge-based model. This is because the cloud-based model can utilize the complete and complex training dataset, rather than a limited subset of the data available to each edge node. Thus, the accuracy increases faster than the edge-based model. In the second part, the workers have trained enough datasets and uploaded the parameters to the server. Further, the different workers are fed by different subsets of the total dataset which is equivalent to data sampling, and it obtains better performance than the cloud-based model in this stage. During the steps 2000 to 3000, the two models show the same performance, matching the theoretical analysis in Section IV-B4.

## B. Proposed CNN Model vs. Existing CNN Models

After the evaluation between cloud-based and edge-based models, we deploy several existing CNN models to the edge nodes and compare their performance. We utilize existing code implementations [36] and modify the code with TensorFlow distribution methods. Thus, all the existing CNN models are implemented in the distributed manner in order to make a fair comparison between the proposed CNN models. Fig. 10 illustrates the cross-entropy loss for each CNN model. The losses for all the models have similar patterns. The losses decrease rapidly before the $500^{th}$ step and are stable after the $500^{th}$ step. The results illustrate that the proposed CNN model achieves the same loss performance compared with the existing CNN models. Furthermore, Fig. 11 illustrates the classification accuracy for all the CNN models. The performance of the proposed CNN and the VGG-16 models reach approximately 95% accuracy, while the LeNet-5 model reaches 89.6% accuracy. It clearly confirms that the proposed CNN model and the VGG-16 model have similar classification accuracy and their performance is better than the LeNet-5 model.

## C. Training Time Consumption Comparison

We also compare the training times for different CNN models. Fig. 12 illustrates the time costs of four CNN models: the proposed edge-based model, the VGG-16 model [35], the LeNet5 model [30], and the cloud-based model from the training process. Here, the $x$-axis represents the different CNN models and the $y$-axis indicates the total time cost in minutes. Further, each model is executed 10 times and we utilize 95% confidence intervals. The evaluation results show that the training time cost for the cloud-based model is over 85 minutes, while the edge-based model is less than 35 minutes, approximately 38% of the time cost for cloud-based model. Meanwhile, we obtain the time cost for the distributed LeNet5 and VGG-16 models as well. From the figure, we observe that our model benefits from the parameter optimization on this specific dataset, as our proposed model achieves the smallest computation time in the convolutional layers to abstract the features, leading to the shortest computation time of all the schemes.

To summarize, our proposed edge-based CNN model can not only reduce the training time, but also maintain equivalent performance, compared to the existing CNN models. Furthermore, as discussed in Sections IV-C and IV-B, reducing the output feature size could improve the network performance. Thus, the proposed edge-based CNN model reduces both

network traffic overhead and CNN training time simultaneously, and our proposed CNN model improves the system performance for our IIoT scenario.

## VII. Discussion

In this study, we propose an edge-based CNN model which is deployed on edge nodes to offload the training process from the cloud to the edge, thereby avoiding the data exchanges between servers and IoT devices and reducing network traffic in IIoT. As possible extensions of our work, we consider possible future directions toward improving IIoT with respect to extending learning models, security concerns, and the codesign of control, networking, computing, and learning.

### Learning Model Extension:

In this paper, we have validated that the proposed edge-based CNN model achieves better performance than existing CNN models on the particular T-Less dataset. However, the applicability and extensibility of the proposed model remains unexplored. Generally speaking, as the deep learning process is a black-box process, a deep learning model is generally configured and optimized only for the specific training dataset that it was trained on. When the training dataset changes, the system has to reconfigure, or retrain, the deep learning model to obtain accurate results. Thus, how to design a generalized learning model to handle different datasets and achieve accurate results remains a challenging problem. As ongoing research, we plan to extend our work to apply different types of datasets to our proposed model, and design a generic learning model to adopt multiple datasets.

As IIoT is a dynamic system, the system generates new data constantly over time. Thus, the CNN model needs to retrain to maintain model accuracy when the system receives new data. The training cost increases constantly, since the size of the training dataset increases constantly. To handle this, an online learning strategy should be considered. Online learning utilizes model updating instead of retraining to process the new data, which means the model only utilizes the new datasets to train the model instead of utilizing the entire dataset (including old and new data), which reduces the training time. Thus, it is possible to utilize the online learning strategy to optimize the proposed edge-based CNN model to adapt to the dynamic IIoT environment. Specifically, the parameter server maintains and manages the well-trained model. Also, it defines a suitable threshold for the classification accuracy. As the system changes dynamically, such as through the addition of new parameters or new productions in the system, the accuracy of the old model will certainly drop. When the accuracy drops below the threshold, the edge workers should update the edge-based model by training with the newly generated datasets. To do so, we can reduce the learning overhead and increase the flexibility of the deep learning model for the dynamics system.

### IIoT Security:

The security of the IIoT system is important, how to leverage system identification, vulnerability analysis, and resilience operations to mitigate the network risk is one critical issue to be addressed [37], [38], [39], [40]. In particular, as mentioned above, the automation and intelligence of IIoT is based upon the efficacy of data analysis. Compromising the data

analysis causes the system to instigate improper control and obtain unexpected results. To this end, there are credible threats to the data analysis process. One possible mechanism is that an adversary could tamper with the labels of the datasets affect the training process and yield incorrect analysis results when the model is deployed [41]. Furthermore, the adversary could tamper with the parameters for the deep learning model, which also affects the training results [42]. Since the deep learning process is a black box process, it requires extensive experiments to determine the level of impact of different threats on IIoT by involving a set of metrics to conduct systematic risk assessment. Based on such experiments, how to design effective defensive techniques to protect the deep learning process remains another challenging issue, which is another research direction.

To mitigate the risks of attacks, we consider several possible solutions. First, it is critical to improve the security of machine learning algorithms and models. Indeed, existing research efforts have shown that adversarial learning can compromise most machine learning models, including CNNs, DNNs, and others [43], [44], [45]. Thus, strengthening the security of machine learning itself is a necessary goal. Second, because of the wide use of machine learning in IIoT, the security risks of machine learning inevitably affect the security of IIoT systems. Adversaries could launch attacks against machine learning algorithms deployed in IIoT systems so that the performance of IoT systems could be reduced or altered. As a typical distributed system, data in IIoT is collected by sensors in different locations and transmitted to servers for further analysis. We categorize the data collection process into three phases: data collection, transmission, and processing. In fact, adversaries could launch attacks against any or all phases (e.g., injecting false data in the data collection phase). Thus, an effective end-to-end defense solution must be designed to not only protect machine learning models and mechanisms, but also to protect the data in the collection, transmission, and storage processes. It is also necessary to develop recovery mechanisms to restore IIoT systems when they are under attack or compromised.

### Co-design of Control, Networking, Computing, and Learning:

The control, networking, and data analysis are the key components of IIoT [1]. This is equally true for IIoT, which presents its own unique challenges and opportunities. Thus, how to leverage these three components in IIoT systems to optimize the existing industrial model is critical. The control system in an IIoT environment plays a crucial role in controlling and operating critical infrastructures, which not only requires the network system support to transmit the control signal, but also requires data analysis support to make correct decisions and increase the control accuracy. Indeed, designing the three sub-systems of control, communication, and data analysis independently creates problems. For instance, the control system may be operated manually by the manager/controller according to the data analysis results, which is a gap in the IIoT automation closed loop. Further, the network system takes the responsibility of transmitting the datasets and control signals, and the performance of the network system directly impacts the performance of every other system. Thus, it is necessary to co-design these subsystems to interoperate cohesively.

Nonetheless, many critical issues are still open at this stage. From the control point of view, how to manage and control various facilities is a challenging problem. First, recall that

IoT provides ubiquitous connections, and thus integration of control hardware and software is difficult. Also, the control system is time sensitive, and how to guarantee the control signal is timely and accurate is challenging. From the network point of view, the existing network protocols do not fit low-power IoT devices. Thus, how to integrate new network technologies such as 5G [46], [47], machine-to-machine (M2M) communications [48], [49], and SDN [50] with IoT remains an open issue. Another challenge is network deployment, which includes identification, network structure, consideration for densification, distribution, and mobility. Finally, how to improve the efficiency of the data analysis and how to reduce the computation requirements of the analysis process are critical issues.

In an IIoT system, edge computing, as a new distributed computing paradigm, can offload computation tasks from computing centers to the network edge, so that the latency of transmitting data collected from sensors to data analysis components can be reduced, as edge nodes are much closer to sensors. In the same way, decisions can be quickly delivered from edge nodes to actuators. To make accurate and rapid decisions, machine learning can be deployed at edge nodes. Nonetheless, edge nodes have limited computation resources, it is necessary to design cost-effective machine learning schemes that can support edge computing-based data analytics to aid decisionmaking in IIoT systems. While machine learning has achieved great success in a number of applications, such as image/video recognition, natural language process, and others, the design of machine learning techniques that can deal with the exceptional requirements of IIoT systems in terms of safety, accuracy, and real-time response must be realized.

## VIII. Final Remarks

In this paper, we formalized the problem space for IIoT in network, computation, and structure, and focused on the offloading of the deep learning from cloud servers to edge nodes in order to avoid the massive amount of data exchanged between servers and IoT devices. Based on our problem formalization, we proposed an edge-based CNN model, which moves the CNN model used to classify the manufacturing components in IIoT, from the cloud servers to the edge nodes. To deploy the proposed CNN model to the edge nodes, we optimized the parameters of the CNN model to reduce the training time and computation time of the model. Based on the system model, we analyzed the performance between the cloud-based CNN model and our proposed edge-based model. To evaluate the proposed CNN model, we designed a comprehensive simulation based on the Google Cloud Instance. We also created an edge computing testbed on Google Cloud and deployed the proposed model to the testbed. Extensive experimental results indicate that our proposed edge-based CNN model is capable of not only offloading the computation to avoid massive and costly data exchanges between cloud servers and IoT devices, but also reducing the training time while obtaining the similar classification accuracy comparing to several baseline schemes.

## References

[1]. Xu H, Yu W, Griffith D, and Golmie N, “A survey on industrial internet of things: A cyber-physical systems perspective,” IEEE Access, vol. 6, pp. 78238–78259, 2018.

[2]. Jeschke S, Brecher C, Song H, and Rawat DB, Industrial Internet of Things: Cybermanufacturing Systems. Springer, 2017.

[3]. Dartmann G, Song H, and Schmeink A, Big Data Analytics for Cyber-Physical Systems: Machine Learning for the Internet of Things. Elsevier, 2019.

[4]. Liang F, Yu W, An D, Yang Q, Fu X, and Zhao W, "A survey on big data market: Pricing, trading and protection," IEEE Access, vol. 6, pp. 15132–15154, 2018.

[5]. Yu W, Liang F, He X, Hatcher WG, Lu C, Lin J, and Yang X, "A survey on the edge computing for the internet of things," IEEE access, vol. 6, pp. 6900–6919, 2018.

[6]. Shi W, Cao J, Zhang Q, Li Y, and Xu L, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp.637–646, Oct 2016.

[7]. Jeschke S, Brecher C, Meisen T, zdemir D, and Eschert T, Industrial Internet of Things and Cyber Manufacturing Systems, ser. Springer Series in Wireless Technology. Cham: Springer International Publishing, 2016, pp. 3–19. [Online]. Available: https://publications.rwth-aachen.de/record/689897

[8]. Hatcher WG and Yu W, "A survey of deep learning: Platforms, applications and emerging research trends," IEEE Access, vol. 6, pp. 24411–24432, 2018.

[9]. Mohammadi M, Al-Fuqaha A, Sorour S, and Guizani M, "Deep learning for iot big data and streaming analytics: A survey," IEEE Communications Surveys Tutorials, vol. 20, no. 4, pp. 2923–2960, Fourthquarter 2018.

[10]. Boschert S and Rosen R, "Digital twinthe simulation aspect," in Mechatronic Futures. Springer, 2016, pp. 59–74.

[11]. Qi Q and Tao F, "Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison," IEEE Access, vol. 6, pp. 3585–3593, 2018.

[12]. Canedo A, "Industrial iot lifecycle via digital twins," in 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS). IEEE, 2016, pp. 1–1.

[13]. Tao F, Sui F, Liu A, Qi Q, Zhang M, Song B, Guo Z, Lu SC-Y, and Nee A, "Digital twin-driven product design framework," International Journal of Production Research, pp. 1–19, 2018.

[14]. Thoben K-D, Wiesner S, and Wuest T, "industrie 4.0 and smart manufacturing-a review of research issues and application examples," International Journal of Automation Technology, vol. 11, no. 1, pp. 4–16, 2017.

[15]. Peralta G, Iglesias-Urkia M, Barcelo M, Gomez R, Moran A, and Bilbao J, "Fog computing based efficient iot scheme for the industry 4.0," in 2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM). IEEE, 2017, pp. 1–6.

[16]. Li X, Li D, Wan J, Liu C, and Imran M, "Adaptive transmission optimization in sdn-based industrial internet of things with edge computing," IEEE Internet of Things Journal, vol. 5, no. 3, pp. 1351–1360, 2018.

[17]. Xu X, "From cloud computing to cloud manufacturing," Robotics and computer-integrated manufacturing, vol. 28, no. 1, pp. 75–86, 2012.

[18]. Jiang C, Zhang H, Ren Y, Han Z, Chen K-C, and Hanzo L, "Machine learning paradigms for next-generation wireless networks," IEEE Wireless Communications, vol. 24, no. 2, pp. 98–105, 2017.

[19]. Zhu J, Song Y, Jiang D, and Song H, "A new deep-q-learning-based transmission scheduling mechanism for the cognitive internet of things," IEEE Internet of Things Journal, vol. 5, no. 4, pp. 2375–2385, 2018.

[20]. Mocanu DC, Mocanu E, Nguyen PH, Gibescu M, and Liotta A, "Big iot data mining for real-time energy disaggregation in buildings," in 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2016, pp. 003765–003769.

[21]. Huang Y, Ma X, Fan X, Liu J, and Gong W, "When deep learning meets edge computing," in 2017 IEEE 25th International Conference on Network Protocols (ICNP). IEEE, 2017, pp. 1–2.

[22]. Zhang Q, Yang LT, Chen Z, and Li P, "A tensor-train deep computation model for industry informatics big data feature learning," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3197– 3204, 2018.

[23]. Zhang Q, Yang LT, Chen Z, Li P, and Bu F, "An adaptive dropout deep computation model for industrial iot big data learning with crowdsourcing to cloud computing," IEEE Transactions on Industrial Informatics, vol. 15, no. 4, pp. 2330–2337, 2018.

[24]. Ghobakhloo M, “The future of manufacturing industry: A strategic roadmap toward industry 4.0,” Journal of Manufacturing Technology Management, vol. 29, no. 6, pp. 910–936, 2018.

[25]. LeCun Y, Bengio Y, and Hinton G, “Deep learning,” nature, vol. 521, no. 7553, p. 436, 2015. [PubMed: 26017442]

[26]. Hodaň T, Haluza P, Obdržálek Š, Matas J, Lourakis M, and Zabulis X, “T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects,” IEEE Winter Conference on Applications of Computer Vision (WACV), 2017.

[27]. Gidaris S and Komodakis N, “Object detection via a multi-region and semantic segmentation-aware cnn model,” in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1134–1142.

[28]. Lipton ZC, “The mythos of model interpretability,” arXiv preprint arXiv:1606.03490, 2016.

[29]. Sze V, Chen Y-H, Yang T-J, and Emer JS, “Efficient processing of deep neural networks: A tutorial and survey,” Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, 2017.

[30]. El-Sawy A, Hazem E-B, and Loey M, “Cnn for handwritten arabic digits recognition based on lenet-5,” in International Conference on Advanced Intelligent Systems and Informatics. Springer, 2016, pp. 566–575.

[31]. Qassim H, Verma A, and Feinzimer D, “Compressed residual-vgg16 cnn model for big data places image recognition,” in 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2018, pp. 169–175.

[32]. Zhang C, Wu D, Sun J, Sun G, Luo G, and Cong J, “Energy-efficient cnn implementation on a deeply pipelined fpga cluster,” in Proceedings of the 2016 International Symposium on Low Power Electronics and Design. ACM, 2016, pp. 326–331.

[33]. Zhao W, Fu H, Luk W, Yu T, Wang S, Feng B, Ma Y, and Yang G, “F-cnn An fpga-based framework for training convolutional neural networks,” in 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 2016, pp. 107–114.

[34]. Google Cloud, https://cloud.google.com/.

[35]. Han S, Pool J, Tran J, and Dally W, “Learning both weights and connections for efficient neural network,” in Advances in neural information processing systems, 2015, pp. 1135–1143.

[36]. Deep Learning Documents, http://deeplearning.net/tutorial/lenet.html/.

[37]. Liu X, Qian C, Hatcher WG, Xu H, Liao W, and Yu W, “Secure internet of things (iot)-based smart-world critical infrastructures: Survey, case study and research opportunities,” IEEE Access, pp. 1–1, 2019.

[38]. Lin J, Yu W, Zhang N, Yang X, Zhang H, and Zhao W, “A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications,” IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1125–1142, Oct 2017.

[39]. Tuptuk N and Hailes S, “Security of smart manufacturing systems,” Journal of Manufacturing Systems, vol. 47, pp. 93–106, April 2018.

[40]. Ling Z, Luo J, Xu Y, Gao C, Wu K, and Fu X, “Security vulnerabilities of internet of things: A case study of the smart plug system,” IEEE Internet of Things Journal, vol. 4, no. 6, pp. 1899–1909, Dec 2017.

[41]. Yu D and Deng L, “Deep learning and its applications to signal and information processing [exploratory dsp],” IEEE Signal Processing Magazine, vol. 28, no. 1, pp. 145–154, 2010.

[42]. Hu W and Tan Y, “Generating adversarial malware examples for black-box attacks based on gan,” arXiv preprint arXiv:1702.05983, 2017.

[43]. Tzeng E, Hoffman J, Saenko K, and Darrell T, “Adversarial discriminative domain adaptation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 7167–7176.

[44]. Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, and Swami A, “The limitations of deep learning in adversarial settings,” in 2016 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2016, pp. 372–387.

[45]. Liang F, Hatcher WG, Liao W, Gao W, and Yu W, “Machine learning for security and the internet of things: The good, the bad, and the ugly,” IEEE Access, vol. 7, pp. 158126–158147, 2019.

[46]. Agiwal M, Roy A, and Saxena N, “Next generation 5g wireless networks: A comprehensive survey,” IEEE Communications Surveys Tutorials, vol. 18, no. 3, pp. 1617–1655, thirdquarter 2016.

[47]. Yu W, Xu H, Zhang H, Griffith D, and Golmie N, “Ultra-dense networks: Survey of state of the art and future directions,” in 2016 25th International Conference on Computer Communication and Networks (ICCCN), Aug 2016, pp. 1–10.

[48]. Wu Y, Yu W, Griffith DW, and Golmie N, “Modeling and performance assessment of dynamic rate adaptation for m2m communications,” IEEE Transactions on Network Science and Engineering, pp. 1–1, 2018.

[49]. Kim J, Lee J, Kim J, and Yun J, “M2m service platforms: Survey, issues, and enabling technologies,” IEEE Communications Surveys Tutorials, vol. 16, no. 1, pp. 61–76, First 2014.

[50]. Kreutz D, Ramos FMV, Verssimo PE, Rothenberg CE, Azodolmolky S, and Uhlig S, “Software-defined networking: A comprehensive survey,” Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, Jan 2015.

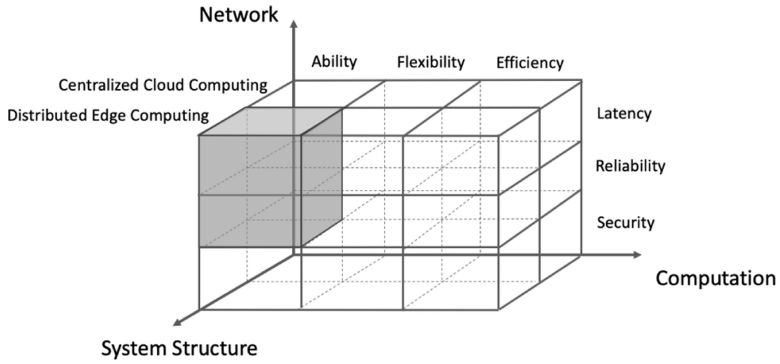


**Fig. 1.**
The Problem Space of the IIoT

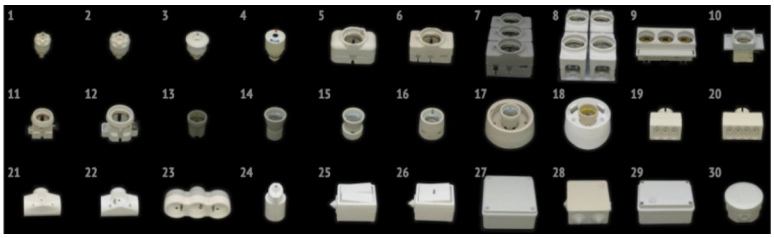


**Fig. 2.**
Examples of the T-Less Dataset

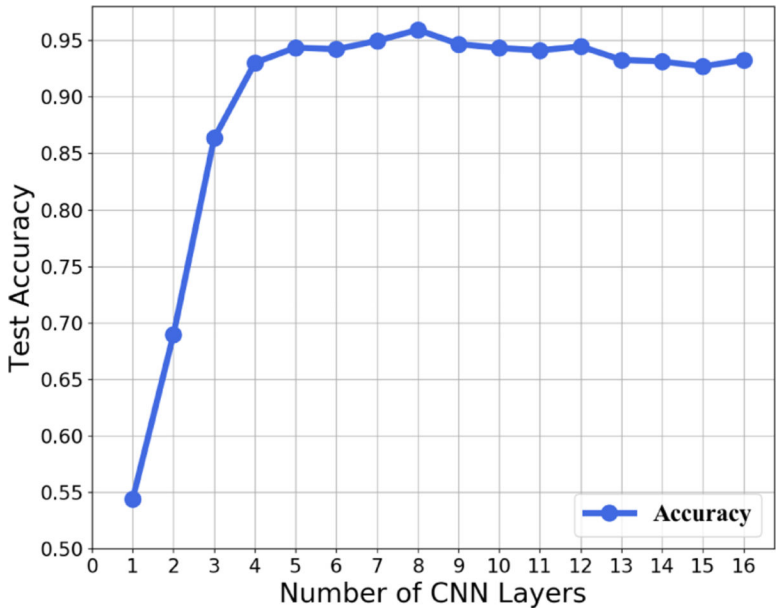


**Fig. 3.**
Test Accuracy for Different Number of CNN Layers

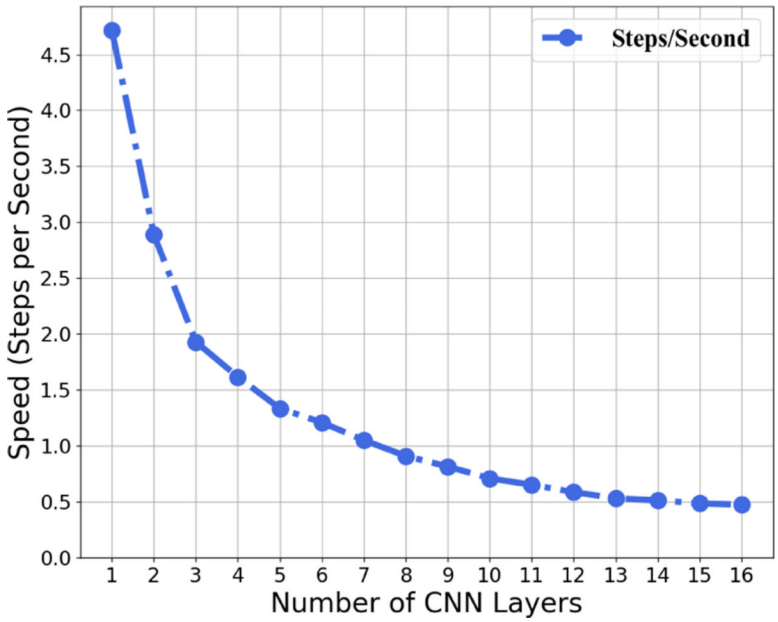


**Fig. 4.**
Training Speed for Different Number of CNN Layers

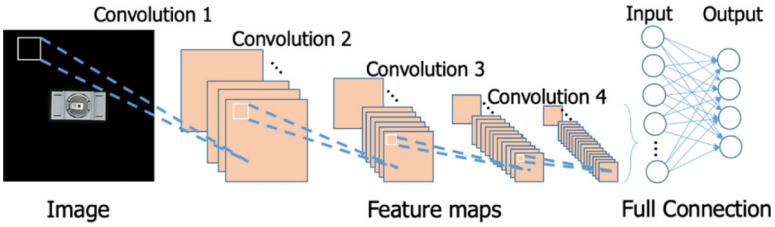


**Fig. 5.**
Structure of CNN Model

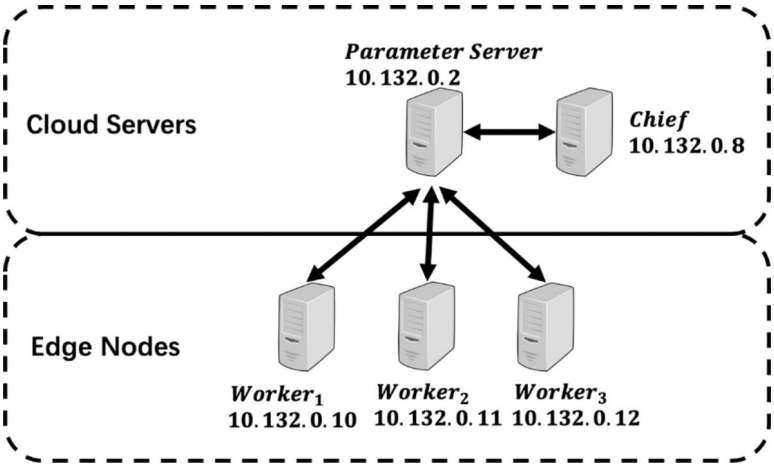


**Fig. 6.**
Testbed Structure

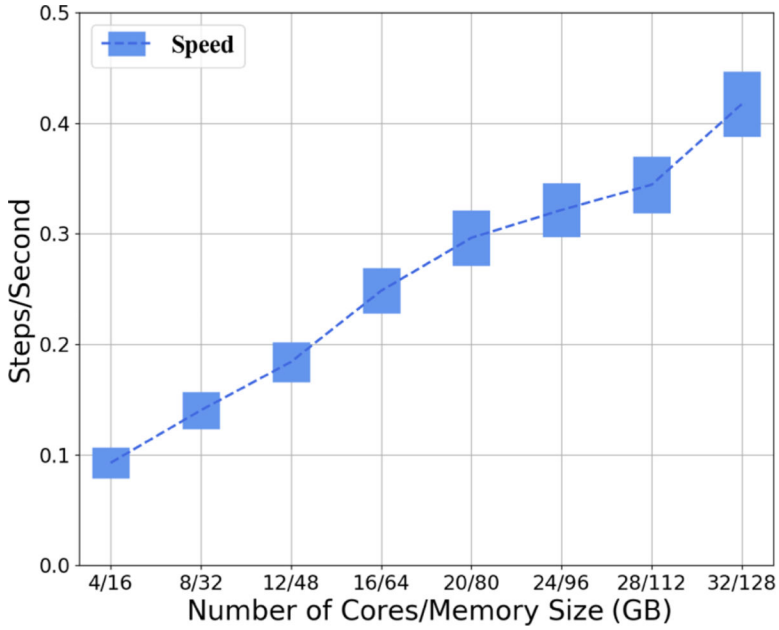


**Fig. 7.**
Computation Power Estimation, with 95% confidence intervals shown

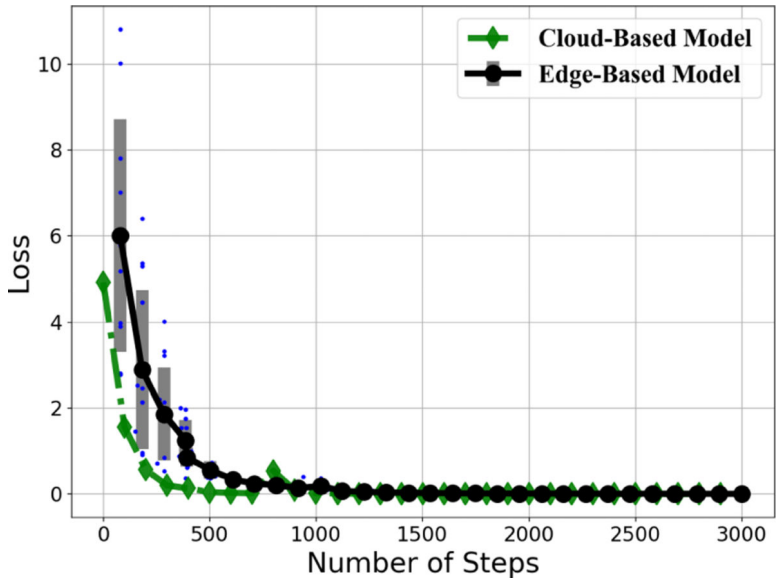


**Fig. 8.**

Loss Comparison Between Cloud-Based and Edge-Based Model, with 95% confidence intervals shown

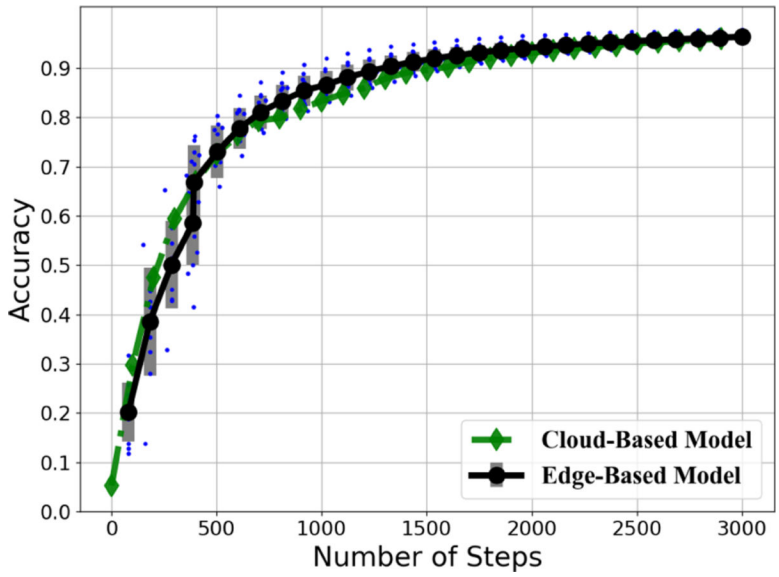


**Fig. 9.**

Accuracy Comparison Between Cloud-Based and Edge-Based Model, with 95% confidence intervals shown

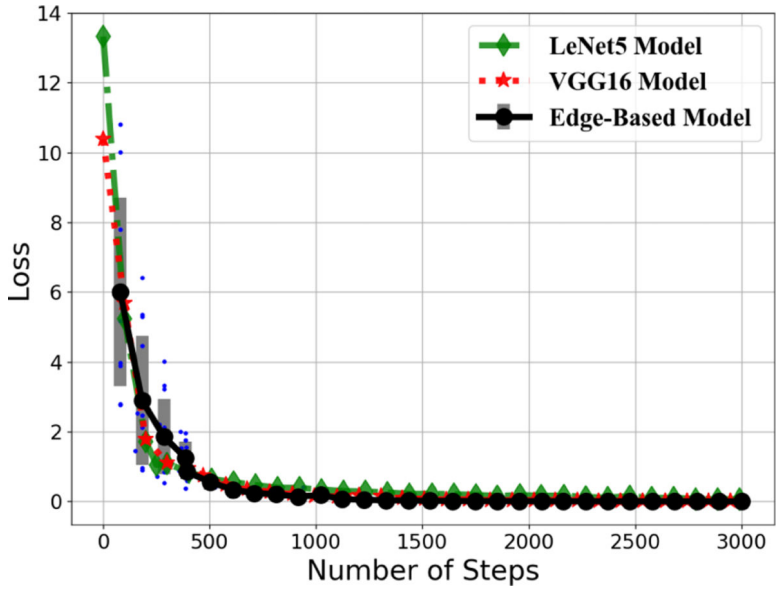


**Fig. 10.**

Loss Comparison for Different CNN Models, with 95% confidence intervals shown

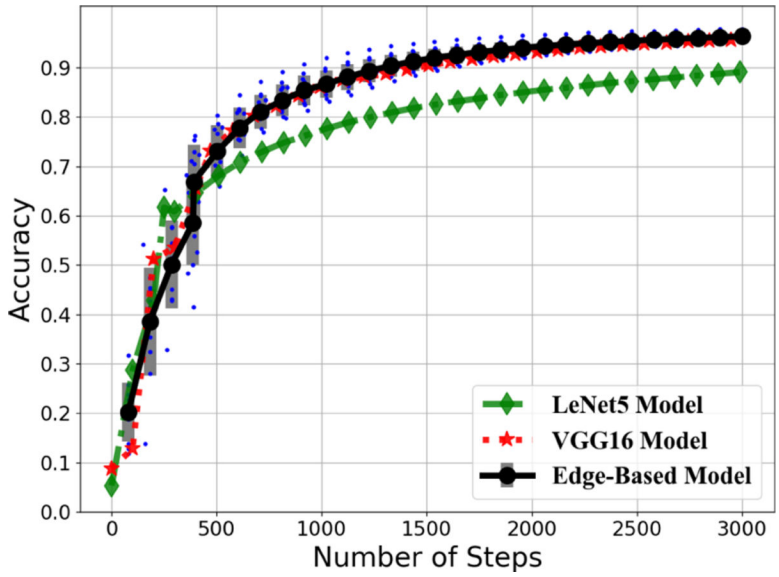


**Fig. 11.**

Accuracy Comparison for Different CNN Models, with 95% confidence intervals shown

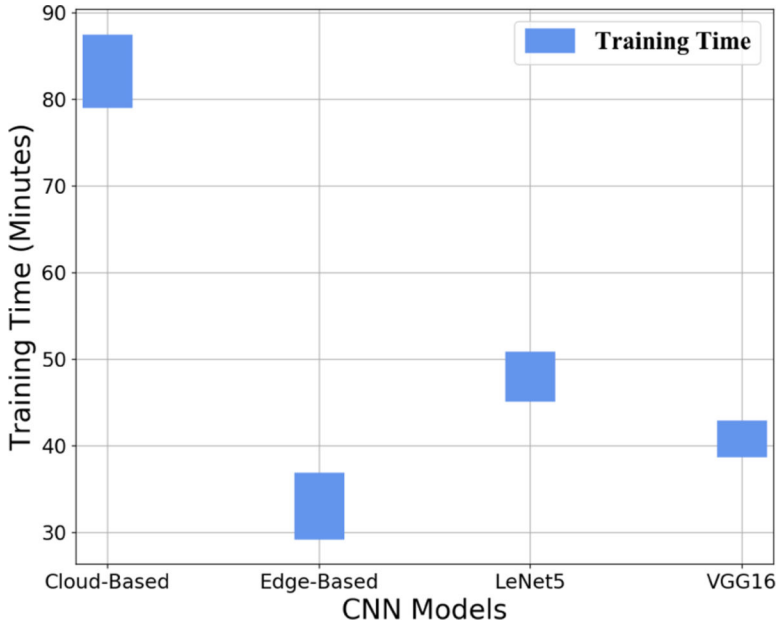


**Fig. 12.**
Training Time for Different CNN Models, with 95% confidence intervals shown

**TABLE I**

NOTATIONS

| Symbols | Descriptions |
|---|---|
| $T$ | Computation time for CNN model |
| $c$ | Number of image channels |
| $m$ | Number of convolutional filters |
| $\mathbb{C}$ | Results from prior convolutional layer |
| $\mathbb{I}, \mathbb{K}$ | Input of CNN model and convolution kernel |
| $P$ | Pooling result |
| $z, w, b$ | Forward propagation result, weight, and bias |
| $\delta_{n-1}$ | Backward propagation result for layer $n-1$ |
| $Loss_j$ | The loss value after layer $j$ |
| $\alpha$ | Learning rate |
| $T_d$ | System delay for cloud-based model |
| $T_c$ | Data collection time |
| $T_{t1}$ | Transmission time (sensors to edge nodes) |
| $T_r$ | Data receiving time for edge nodes |
| $T_{t2}$ | Transmission time (edge nodes to servers) |
| $T_p$ | Data analysis time for cloud-based model |
| $D_s$ | Data size collected by each sensor |
| $D_e$ | Data size received by each edge node |
| $T_d^e$ | System delay for edge-based model |
| $T_p^e$ | Data analytical time for edge-based model |
| $T_r^e$ | Data receiving time for the server |

**TABLE II**

LEARNING RATE IDENTIFICATION

| Learning Rate | Training Steps | Training Loss |
|---|---|---|
| **0.05** | 15 | NaN |
| **0.01** | 107 | NaN |
| **0.005** | 1500 | 0.086 |
| **0.004** | 1500 | 0.083 |
| **0.003** | 1500 | 0.081 |
| **0.002** | 1500 | 0.073 |
| **0.001** | 1500 | 0.068 |