A Cloud-Edge Collaboration Framework for Cognitive Service

Chuntao Ding, Ao Zhou, Yunxin Liu, *Senior Member, IEEE,* Rong N. Chang, *Senior Member, IEEE,* Ching-Hsien Hsu, *Senior Member, IEEE,* Shangguang Wang, *Senior Member, IEEE*

Abstract—Mobile applications can leverage high-quality deep learning models such as convolutional neural networks and deep neural networks to provide high-performance cognitive services. Prior work on deep learning models-based mobile applications in a cloud-edge computing environment focuses on performing lightweight data pre-processing tasks on edge servers for cloud-hosted cognitive servers. These approaches have two major limitations. First, it is uneasy for the mobile applications to assure satisfactory user experience in terms of network communication delay, because the intermediary edge servers are used only to pre-process data (*e.g.*, images and videos) and the cloud servers are used to complete the tasks. Second, these approaches assume the pre-trained deep learning models deployed on cloud servers are static, and will not attempt to automatically upgrade in a context-aware manner. In this paper, we propose a cloud-edge collaboration framework that facilitates delivering cognitive services with long-lasting, fast response, and high accuracy properties. We fist deploy a shallow model (i.e., EdgeCNN) on the edge server and a deep model (i.e., CloudCNN) on the cloud server. EdgeCNN can provide durable and rapid response cognitive services, because edge servers not only provide computing resources for mobile applications, but also close to users. Then, we enable CloudCNN to assist in training EdgeCNN to improve the performance of the latter. Thus, EdgeCNN also provides high-accuracy cognitive services. Furthermore, because users may continue to upload data to edge servers in real-world scenarios, we propose to use the ongoing assistance of CloudCNN to further improve the accuracy of the shallow model. Experimental results show that EdgeCNN can reduce the average response time of cognitive services by up to 55.08% and improve accuracy by up to 26.70%.

Index Terms—Cognitive service, Cloud-Edge collaboration, cloud computing.

1 Introduction

1.1 Motivation & Problem Formulation

Deep Learning Models (DLMs) have recently enjoyed a great success in cognitive services [1], [2]. At the same time, we are witnessing the explosive growth of mobile devices. Deploying DLMs on such devices will definitely make our lives easier by providing high-quality cognitive services. However, high-performance DLMs often require a large amount of computing and storage resources, which makes them difficult to use in resource-limited mobile devices. To address this problem, one solution is to use the mobile cloud computing architecture [3], [4], *i.e.*, the user first uploads the captured data to the cloud, and then it is processed by the DLMs deployed on the cloud to obtain results. Nevertheless, uploading the captured data to the cloud may cause long transmission delays because users are usually far away from the cloud. The other solution is to deploy DLMs on mobile devices. However, it is difficult to make mobile devices provide long duration cognitive services because running DLMs can quickly drain their battery. As a matter of fact, many cognitive services require quick response and long duration. For instance, older people wear Google Glass to navigate through the road, and people use Hololens 2 to learn and collaborate on tasks. Mobile edge computing [5]–[12] offers a new solution motif in which mobile devices can offload partial or all of their tasks to edge servers located at the edge of the network. Because edge servers are deployed near users, it is technically feasible to provide users with cognitive services through edge servers.

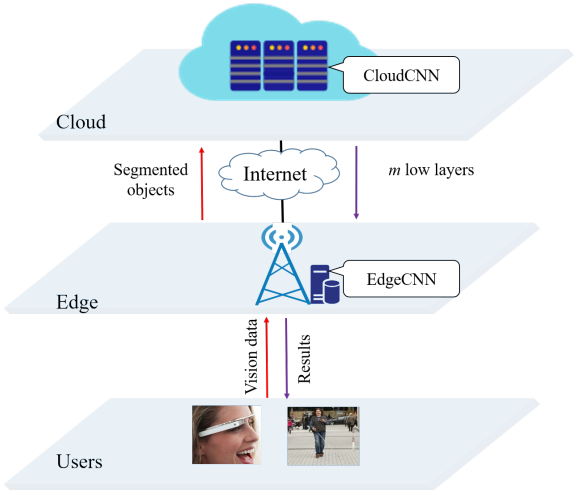


Fig. 1. Architecture of the proposed framework.

This paper addresses the problem of cognitive services in the mobile edge computing paradigm, which is formulated as follows. As illustrated in Fig. 1, the system architecture consists of three layers: users, edge, and cloud. A small amount of labeled data is stored on the edge server and

- *C. Ding, A. Zhou, S. Wang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876. E-mail: {ctding; aozhou; sgwang}@bupt.edu.cn.*
- *Y. Liu is with Microsoft Research, Beijing 100080 E-mail: yunxin.liu@microsoft.com.*
- *R. Chang is with IBM T. J. Watson Research Center, Hawthorne, NY 10532. E-mail: rong@us.ibm.com.*
- *C. Hsu is with the College of Information and Electrical Engineering, Asia University, Taiwan. E-mail:robertchh@gmail.com*

denoted as $\{(x_i, y_i)\}_{i=1}^{N}$, where N is the number of data, $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}^c$, d is the dimension of x_i, c is the number of classes. This paper takes convolutional neural networks as an example. We deploy the convolutional neural network model on the edge server to learn f_e, where $f_e(\mathbf{W}_e) = \arg\min_{f \in \mathcal{F}_e} \frac{1}{N} \sum_{i=1}^{N} \mathcal{H}(y_i, \sigma(f(x_i; \mathbf{W}_e)))$, and $\mathcal{F}_e$ is a class of function from $\mathbb{R}^d$ to $\mathbb{R}^c$. The function σ is the softmax operation, where $\sigma(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{c} e^{z_j}}$, and $1 \leq k \leq c$. The function $\mathcal{H}$ is the cross entropy loss $\mathcal{H}(y, \widehat{y}) = -\sum_{i=1}^{c} y_i log \widehat{y}_i$, where y_i is the true label of x_i, $\widehat{y}_i$ is x_i's predicted label, and $\mathbf{W}_e$ denotes parameters of the model. This paper studies how to learn f_e to improve the accuracy of the model deployed on the edge server.

1.2 Limitations of Prior Art

Related studies on providing cognitive services based on mobile edge computing are aimed at reducing response time and overloading of the core network [2], [13], [14]. However, they have two major limitations. First, it is uneasy to assure satisfactory user experience in terms of network communication delay, because the intermediary edge servers are used only to pre-process the data. Second, these approaches assume that the pre-trained DLMs deployed on cloud servers will not change, nor will it attempt to automatically improve model accuracy in a context-aware manner.

1.3 Proposed Approach

In this paper, we propose a cloud-edge collaboration framework for cognitive services. Fig. 1 illustrates the architecture of the proposed framework. We deploy a shallow model (called EdgeCNN) on the edge server. The EdgeCNN can meet the use's fast response requirements by reducing multiplication operations when inferring tasks. For instance, let $\Phi_j \in \mathbb{R}^{w_j \times h_j \times c_j}$ denote input feature maps of the $(j+1)$th convolutional layer, where w_j and h_j are the width and height of each input feature map; and c_j is the number of input feature maps. The $(j+1)$th layer consists of c_{j+1} filters with size $f_w \times f_h \times c_j$ (where f_w and f_h are the spatial size of the filter). It applies these filters onto the input feature maps Φ_j to generate the output feature maps $\Phi_{j+1} \in \mathbb{R}^{w_{j+1} \times h_{j+1} \times c_{j+1}}$. This process involves a total of $c_j f_w f_h c_{j+1} w_{j+1} h_{j+1}$ floating point operations. Thus, the total complexity of all layers is $O(\sum_{j=1}^{m} c_j f_w f_h c_{j+1} w_{j+1} h_{j+1})$, where m is the number of layers. The more layers are, the longer the inferring time is. EdgeCNN has a simple structure and can meet the user's quick response requirements. But an edge server collects only a small amount of data, and training EdgeCNN from scratch can cause over-fitting problems [15] and consumes a lot of computing resources. Inspired by the octopus that its "big brain" and "small brain" can collaborate on complex tasks such as coordinating eight tentacles and hundreds of sensitive suckers [16], [17], we deploy a deep model (called CloudCNN) on the cloud server to assist EdgeCNN to avoid over-fitting problem. We can train CloudCNN on the cloud server for high accuracy because it has rich computing and storage resources and stores a large amount of data. The key idea of this paper is to use CloudCNN to assist in training EdgeCNN and improve the performance of the latter.

In addition, users may continuously upload the captured data to edge servers. We propose to use this data to further improve the performance of EdgeCNN. Therefore, the shallow model can provide faster response and higher accuracy of cognitive services.

1.4 Technical Challenges and Our Solutions

The first technical challenge is how to use CloudCNN to assist in training EdgeCNN to avoid the latter's over-fitting problem [15]. It is difficult to train EdgeCNN with a small amount of data to learn the distribution of the data. In addition, it is challenging because well-trained convolutional neural networks may be privacy-sensitive [18], [19] and cloud servers typically store large amounts of private data, such as medical data. Recently, Yosinski *et al.* [20] verified that lower layer features are more general and can be used as generic image descriptors. Note that the lower layers represent layers closer to the input data, and the other layers are higher layers. Inspired by this, we propose to use the deep convolutional neural network to share its m (where m is a positive integer) lower layers to assist in training the shallow network, as shown in Fig. 1. The CloudCNN first sends its m lower layers to each edge server. On the edge server, some convolutional layers, some fully-connected layers, and one softmax layer are connected to m lower layers to form the EdgeCNN. Then, the EdgeCNN is trained by freezing its m lower layers and fine-tuning remaining layers with the small amount of data. This means that EdgeCNN uses the knowledge of the deep model to assist its training. Thus, EdgeCNN can overcome the over-fitting problem.

The second technical challenge is how to use uploaded data to further improve the accuracy of EdgeCNN. In real-world scenarios, users may continuously upload unlabeled data to the edge server. It is challenging to use unlabeled data to train DLMs. To solve this challenge, we use the deep model to predict the labels of uploaded data since it can achieve high accuracy. That is, when receiving uploaded data, the edge server first pre-processes these data, such as object detection [21] and object segmentation [22]. Then, performing EdgeCNN to get results. In addition, the edge server also saves segmented objects. When the core network load is low, the edge server uploads the segmented objects to the cloud server. When receiving these objects, the cloud server uses CloudCNN to predict the labels and send them to edge servers. Finally, we use the labeled objects to retrain EdgeCNN. This process is ongoing and adaptive. As the number of labeled objects increases, EdgeCNN can further improve its accuracy.

1.5 Novelty and Advantages over Prior Art

The technical novelty of this paper is to propose a cloud-edge collaboration framework. The proposed framework is generic and it can well incorporate most existing high-performance DLMs. The technical depth of this paper is to solve the over-fitting problem of the EdgeCNN and further improve its accuracy by utilizing the uploaded unlabeled data. The key advantages of the proposed framework over previous work are two-fold: (i) EdgeCNN provides faster response cognitive services. This is because the edge server is close to users. In addition, the structure of EdgeCNN

is simpler. (ii) EdgeCNN achieves higher accuracy. This is because it not only uses the shared knowledge of the deep networks, but also retrains itself using the continuously uploaded data. Extensive experimental results show that EdgeCNN can reduce the average response time by 55.08%. Compared with training from scratch, the accuracy of EdgeCNN on CIFAR-10, MNIST and FASHION-MNIST datasets improved by up to 26.70%, 8.76%, and 17.16%. It is worth noting that with the assistance of CloudCNN, the accuracy of EdgeCNN is close to the accuracy of the former.

The following paper is organized as follows. Section 2 reviews related work. Section 3 describes the design of the proposed framework. Section 4 provides the experimental evaluation. Section 5 concludes and outlines future work.

2 Related Work

Cognitive services are a form of service provided by artificial intelligence analysis based on visual input, natural language processing, and speech understanding [23], [24]. In recent years, there has been increasing interest in using deep learning models to improve the performance of mobile applications that provide cognitive services. Existing related work can be divided into three categories: cloud-based [25], device-based [26] and edge-based [13].

The cloud-based mobile application is to run DLMs on cloud servers. That is, users first upload the captured data to cloud servers. When receiving the uploaded data, the cloud servers run DLMs to get results and send the results back to users. For example, Kang *et al.* [25] explored ways to divide deep neural network inference between mobile devices and servers. Fang *et al.* [27] investigated scheduling techniques to improve serving performance in multi-tenant mobile offloading systems. Han *et al.* [28] designed an optimizing compiler and runtime scheduler to systematically trade off deep neural network classification accuracy and remotely service each request. Although these approaches achieve high accuracy, users are far away from the cloud server, which causes long network transmission delays. Therefore, they are not easy to meet the user's rapid response needs.

The device-based mobile application is to directly run DLMs on mobile devices. Compared with cloud servers, mobile devices are constrained by limited computing and storage resources. Unfortunately, DLMs are usually resource-demanding. To enable on-device DLMs, approaches roughly fall into two categories. The first category is to design novel network architectures. For example, Howard *et al.* [26] proposed MobileNet, which uses depthwise separable convolutions to build light weight deep neural networks. Zhang *et al.* [29] proposed ShuffleNet, which utilizes point-wise group convolution and channel shuffle to reduce computation cost while maintaining accuracy. The second category is to use compression techniques to compress DLMs to reduce its resource demands at the expense of accuracy. For example, Hinton *et al.* [30] proposed to transfer knowledge from a teacher network to a student network by learning the class distribution via softened softmax. Fang *et al.* [31] proposed NestDNN, which takes the dynamics of runtime resources into account to satisfy multiple users running CNN models. Akhil *et al.* [32] proposed DeepEye, a match-box sized wearable camera that can run multiple cloud-scale CNN models on the devices and perform rich analysis of the captured data in near real-time. However, the device-based mobile applications have two limitations. First, they will reduce the duration for mobile devices to provide cognitive services. Second, the deployed DLMs are static and cannot be easily updated.

The edge-based mobile application is to provide cognitive services based on mobile edge computing architecture. Drolia *et al.* [13] used the caching model to minimize response time by adaptively balancing load between the edge server and the cloud server. Li *et al.* [14] proposed an edge computing structure for Internet of Things (IoT) deep learning tasks. Authors divide the learning networks into two parts. One part includes lower layers deployed into edge servers, while another part includes higher layers deployed into cloud servers. The edge servers load the intermediate data from the lower layers and then transferred data to the cloud server as the input data for the higher layers. Liu *et al.* [2] split the food recognition tasks between the edge devices and the cloud server. In the proposed system, the edge devices perform light-weight computation on the food images for food recognition. Then, the edge devices upload the pre-processed images to the cloud server for further processing. Other related work that benefits from the mobile edge computing architecture includes [1], [33]. However, they only use edge servers to pre-process data and use cloud servers for further processing. Because users are far away from cloud servers, they are also not easy to provide fast response cognitive services.

3 Design of the Proposed Framework

3.1 Architecture

Fig. 1 illustrates the architecture of the proposed framework, which consists of three layers of components: mobile devices (users), edge servers and cloud servers. The composition and function of each component are introduced as follows:

3.1.1 Mobile Devices

Mobile devices refer to some smart devices with limited computing and storage resources, such as Google Glass, Microsoft Hololens, Smartphone, and Apple Watch. We make mobile devices only responsible for lightweight tasks, such as capturing data, uploading data and receiving results from edge servers to extend their battery life.

3.1.2 Edge Servers

Edge servers are usually a group of servers that are deployed at the edge of the network, such as sensors, home gateways, and micro servers [34]. We use edge servers to perform operations for cognitive services because they are close to users. To this end, we make edge servers responsible for pre-processing the received data, training and performing the shallow models to get results, and uploading the pre-processed data to the cloud servers.

3.1.3 Cloud Servers

Cloud servers are usually AliCloud servers, Amazon Web Service Cloud servers, Microsoft Azure Cloud servers, and Google Cloud servers. Considering that the cloud server has

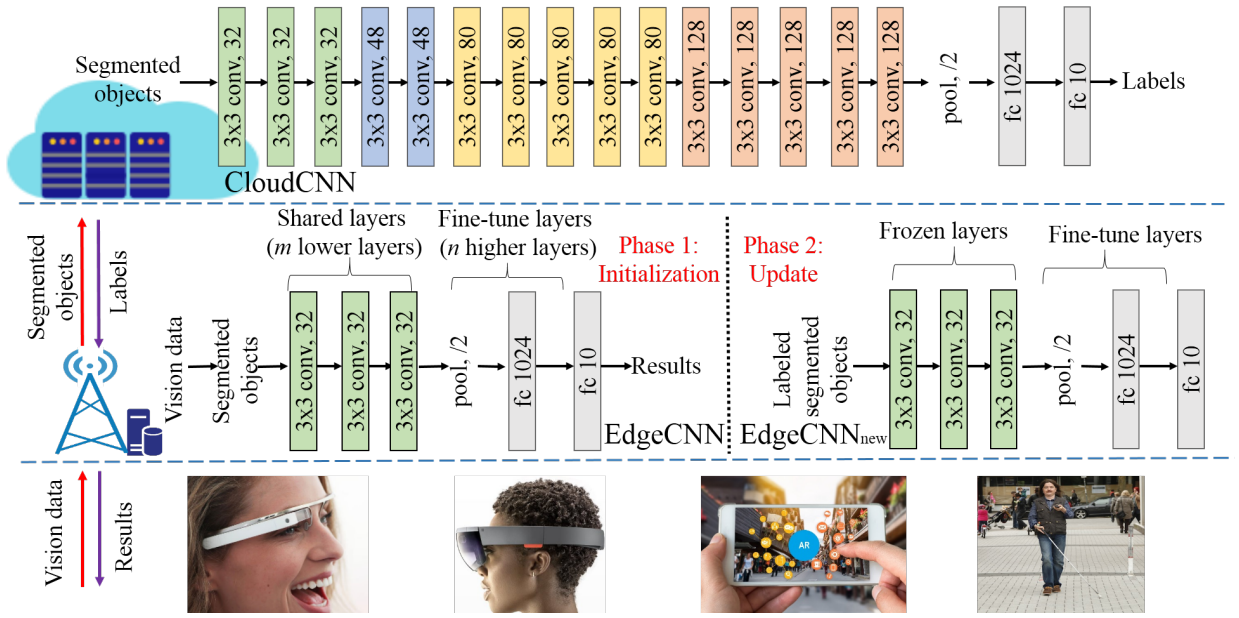


Fig. 2. The detailed process of the proposed framework.

rich computing and storage resources, we make it responsible for computationally intensive tasks, such as training complex models.

In the proposed framework, EdgeCNN improves its performance with the assistance of CloudCNN. However, the interaction between the cloud server and the edge server is transparent to the user, and the user only interacts with the edge server. When a user uses a mobile device to request a cognitive service, the user uploads image data to the nearest edge server. After receiving the data, the edge server first pre-processes it (such as object detection and segmentation), then runs EdgeCNN to get the result and sends it to the user.

3.2 Detailed Process of the Proposed Framework

Fig. 2 illustrates the detailed process of the proposed framework. In this paper, we aim to use edge servers to complete all operations to provide cognitive services, which means that after capturing the data, the mobile device uploads the data directly to the edge server without any preprocessing. This is because mobile devices can reduce their computing load by offloading all operations to the edge server, thereby extending their battery life. In addition, because users are close to the edge server, the transmission delay between them is small.

However, using EdgeCNN directly may get low-accuracy because a single edge server collects only a small amount of data, and training with these data will cause EdgeCNN to overfit. To address this problem, we propose to use the cloud server to assist the edge server. That is, we first train CloudCNN with a large amount of data. As shown in Fig. 2, CloudCNN is deployed on the cloud server, and EdgeCNN is deployed on the edge server. CloudCNN is more complex and has higher accuracy. In order to reduce inference time when providing cognitive services, EdgeCNN has fewer layers. Furthermore, to evaluate the guidance of different layers, we design different layers of EdgeCNN, as shown in Fig. 3. We use CloudCNN to assist in training EdgeCNN by sharing its m lower layers. Thus,

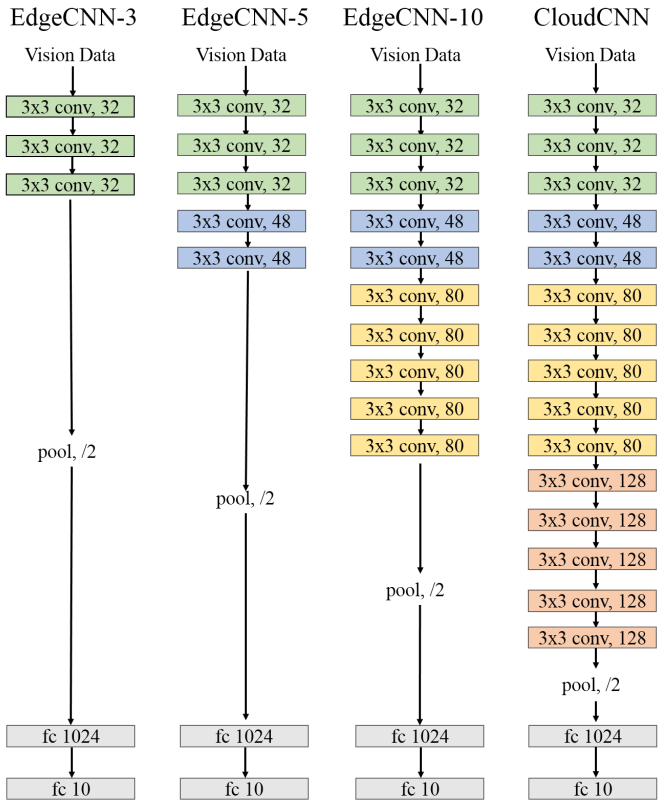


Fig. 3. Example network architectures of CloudCNN and EdgeCNN. Note that, EdgeCNN-3, EdgeCNN-5, and EdgeCNN-10 represent EdgeCNN with 3, 5, and 10 convolution layers, respectively.

the EdgeCNN can avoid training from scratch, which saves a lot of computing resources and improves accuracy.

Furthermore, in reality, users may continue to upload data to edge servers, we propose to use the uploaded data to assist in retraining EdgeCNN to further improve its performance. As shown in Fig. 2, the detailed process of CloudCNN's assistance can be divided into initialization phase and update phase.

3.2.1 Initialization Phase

In the initialization phase, we first use a large amount of data to train CloudCNN on the cloud server. Then, we extract its m lower layers to assist in training of EdgeCNN.

Let $\mathbf{W}_c$ denote parameters of well-trained CloudCNN, $\mathbf{W}_{mc}$ denote parameters of its pre-m lower layers, and $\mathbf{W}_e$ denote parameters of EdgeCNN. With the assistance of CloudCNN, we use $\mathbf{W}_{mc}$ to replace parameters of pre-m lower layers of EdgeCNN. Thus, we only need to train $\mathbf{W}_{e-mc}$, where $\mathbf{W}_{e-mc}$ represents the remaining parameters of EdgeCNN after $\mathbf{W}_{mc}$ is removed. Therefore, given training data $\{x_i, y_i\}_{i=1}^N$, EdgeCNN is trained to optimize the following loss function:

$$f_e(\mathbf{W}_{e-mc}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{H}(y_i, \sigma(f(x_i; \mathbf{W}_{e-mc}))) \quad (1)$$

When the edge server receives the m lower layers (*i.e.*, $\mathbf{W}_{mc}$), the n (where n is a positive integer) higher layers (the parameters are randomly initialized) are connecting to form EdgeCNN, as shown in Fig. 2. After that, EdgeCNN is trained by freezing $\mathbf{W}_{mc}$ and fine-tuning $\mathbf{W}_{e-mc}$, as shown in Eq. 1. Note that the freeze operation means that the frozen parameters will not change when training the neural network model. With the shared m lower layers, EdgeCNN not only improves its accuracy but also saves a lot of computing resources. The reason is that EdgeCNN inherits the knowledge of CloudCNN and only needs to train n higher layers. The detailed initialization process is given in Algorithm 1. Note that, $\mathbf{W}^*_{e-mc}$ represents the updated value of $\mathbf{W}^*_{e-mc}$. Connector $\cup$ represents connecting two parameter sets. For example, S_1 represents the first m-layer parameters of a neural network model, S_2 represents the remaining parameters of this neural network model, and $S_1 \cup S_2$ represents the parameters of the entire neural network model.

Algorithm 1: Initialization Phase

Input: $\{x_i, y_i\}_{i=1}^N$,$\mathbf{W}_c$,$\mathbf{W}_{mc}$, CloudCNN
Output: EdgeCNN
1 CloudCNN sends its m lower layers (*i.e.*, $\mathbf{W}_{mc}$) to edge servers;
2 On each edge server, the n higher layers connect m lower layers to form an EdgeCNN;
3 Initialize $\mathbf{W}_{e-mc}$ to small random values;
4 $\mathbf{W}^*_{e-mc} \leftarrow \arg\min_{\mathbf{W}_{e-mc}} f_e(\mathbf{W}_{e-mc})$;
5 $\mathbf{W}^*_e \leftarrow \mathbf{W}_{mc} \cup \mathbf{W}^*_{e-mc}$;
6 **return** EdgeCNN with $\mathbf{W}^*_e$;

3.2.2 Update Phase

Through the initialization phase, EdgeCNN overcomes its over-fitting problem with the shared layers of CloudCNN. In real-world scenarios, users may continuously upload data to edge servers. In DLMs-based mobile applications, training data is extremely important for improving the performance of mobile applications [35]. Inspired by this, we use uploaded data to further assist in training EdgeCNN. However, this is difficult because these data are unlabeled.

To circumvent this problem, we propose to continue to use CloudCNN. Because CloudCNN is a high-accuracy deep convolutional neural network, such as ResNet [36], which achieves an accuracy of 96.43% on ImageNet. It is applicable to use CloudCNN to predict labels of the uploaded data. In this paper, we assume that CloudCNN always predicts accurate labels. After receiving the uploaded data, the edge server first pre-processes it. For example, we use object detection and object segmentation techniques to obtain segmented objects. This is because object detection and object segmentation can remove some information that is not related to the object, which can reduce the amount of data transmission between the edge server and the cloud server without affecting the object data. Object detection, object segmentation, and object recognition are different convolutional neural network models, and we focus on object recognition. Note that, object detection and segmentation techniques have been well studied in many previous studies. Then, the edge server runs EdgeCNN to get results and return the results to users. It is worth noting that the edge server saves segmented objects. When the core network load is low, the edge server uploads the segmented objects to the cloud server. After receiving the objects, the cloud server uses CloudCNN to predict their labels and sends the labels to edge servers that store corresponding objects. Thus, EdgeCNN can be retrained with those labeled objects. Similar to the initialization phase, EdgeCNN is retrained by freezing its m lower layers and fine-tuning n higher layers.

Let $\mathbf{W}^*_e$ denote parameters of EdgeCNN after the initialization phase. We assume that when the number of newly labeled objects accumulates to M, we start to retrain EdgeCNN. The accumulated objects are denoted as $\{(x_i, y_i)\}_{i=1}^M$. Thus, we use $\{(x_i, y_i)\}_{i=1}^{N+M}$ to retrain EdgeCNN to improve its accuracy. EdgeCNN is retrained by optimizing the following loss function:

$$f_e(\mathbf{W}^{'}_{e-mc}) = \frac{1}{N+M}\sum_{i=1}^{N+M}\mathcal{H}(y_i, \sigma(f(x_i; \mathbf{W}^{'}_{e-mc}))) \quad (2)$$

where $\mathbf{W}^{'}_{e-mc}$ represents the remaining parameters of $\mathbf{W}^*_e$ after $\mathbf{W}_{mc}$ is removed.

CloudCNN can continuously predict labels and send them to the edge server that stores the corresponding data. Therefore, the training process of EdgeCNN is ongoing. The detailed update process is given in Algorithm 2 .

Algorithm 2: Update Phase

Input: EdgeCNN with parameters $\mathbf{W}^*_e$, objects $\{x_i\}_{i=1}^M$, existing visual data set $\{(x_i, y_i)\}_{i=1}^N$, CloudCNN
Output: EdgeCNN$_{new}$
1 When the core network load is low, edge servers upload the visual data $\{x_i\}_{i=1}^M$ to the cloud server;
2 When the cloud server receives $\{x_i\}_{i=1}^M$, they use CloudCNN to get the labels $\{y_i\}_{i=1}^M$;
3 The cloud server sends $\{y_i\}_{i=1}^M$ to edge servers;
4 $\mathbf{W}^{**}_{e-mc} \leftarrow \arg\min_{\mathbf{W}^{'}_{e-mc}} f_e(\mathbf{W}^{'}_{e-mc})$, where $\mathbf{W}^{**}_{e-mc}$ represents the updated value of $\mathbf{W}^{'}_{e-mc}$;
5 $\mathbf{W}^{**}_e \leftarrow \mathbf{W}_{mc} \cup \mathbf{W}^{**}_{e-mc}$;
6 **return** EdgeCNN with $\mathbf{W}^{**}_e$;

In summary, our framework enables the delivery of cognitive services with three benefits: First, the framework

TABLE 1
Statistics of datasets used in the experiments

Dataset	Labels	Training set	Test set
CIFAR-10	10	50,000	10,000
MNIST	10	60,000	10,000
FASHION-MNIST	10	60,000	10,000

can provide long-lasting cognitive services. This is because we shift computing workload from mobile devices to edge servers, reducing the former's computing burden. Second, the framework can provide quick response cognitive services. Since the user is closer to the edge server (usually one hop), the transmission delay between them is small. Furthermore, the EdgeCNN is simple, which also reduces processing time. Therefore, the framework can provide rapid response cognitive services. Third, the framework can provide high-accuracy cognitive services. Through the collaboration of CloudCNN and EdgeCNN and the continuous uploading data, the accuracy of the latter has been significantly improved. It is worth noting the proposed framework is universal, CloudCNN can be any other popular deep neural networks, such as VGG [37] and ResNet [36].

In addition, with the development of the Internet of Things (IoT) and 5G, millions of edge servers will be deployed. Frequent uploading and downloading of data between edge servers and cloud servers will increase the bandwidth burden of the core network. It is worth noting that our framework can alleviate this problem to some extent by uploading only segmented objects and downloading some layers of the model.

4 Experiments

In this section, we first implement a prototype system to verify that EdgeCNN has a fast response. Then, we present the experimental results on three most commonly used datasets (*i.e.*, CIFAR-10 [38], MNIST [39], and FASHION-MNIST [40]) for evaluating visual-related cognitive services. We show that with the assistance of CloudCNN, EdgeCNN can improve its accuracy (*i.e.*, Top-1 accuracy) and shorten training time. In addition, EdgeCNN can further improve its accuracy by using continuously unlabeled data.

4.1 Datasets

We evaluate the proposed framework on three datasets. These datasets are shown in Table 1 and briefly summarized below.

CIFAR-10 [38] is composed of 32×32 RGB images belonging to 10 different classes. The dataset is partitioned into two sets: a training set with 50,000 images and a test set with 10,000 images.

MNIST [39] is a popular dataset for handwritten digit recognition with 10 classes (0-9). The training set contains 60,000 images and the test set contains 10, 000 images. All samples are 28×28 grayscale images.

FASHION-MNIST [40] is composed of 28×28 grayscale images of 70, 000 fashion products from 10 categories, with 7,000 images per category. The training set has 60, 000 images and the test set has 10,000 images.

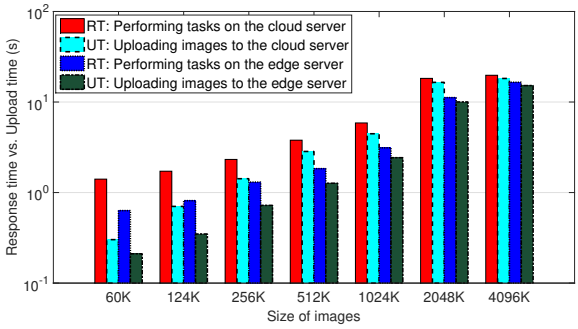


Fig. 4. Response time vs upload time of two different schemes in LTE situation. RT indicates response time, and UT indicates upload time.

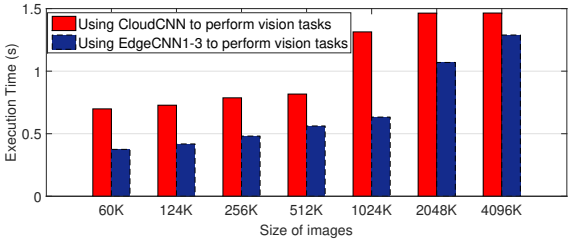


Fig. 5. Execution time of two different schemes.

4.2 Quantitative Mobile Edge Computing Architecture

Compared with the cloud, the edge can provide cognitive services with faster responses. In this section, we implement a prototype system to quantify this. We first deploy the EdgeCNN1-3 (see 4.3.1 for details) on the edge server. After that, we deploy a 19-layer convolutional neural network as CloudCNN on the cloud server. For easy of implementation, CloudCNN and EdgeCNN1-3 contain only one type of convolutional filter. The structures of CloudCNN and EdgeCNN1-3 are shown in Fig. 3.

4.2.1 Experimental Setup

The experiment environment consists of three components: the mobile device, the edge server, and the cloud server. Their configuration is as follows:

Mobile Device: The Huawei Honor 8 smartphone is equipped with 4 Cortex A72 2.3 GHz and Android 7.0 as a mobile device to request cognitive service.

Edge Server: A computer is equipped with an Intel i5-4590@3.3 GHz CPU and 12 GB RAM as an edge server for providing computing and storage resources for tasks.

Cloud Server: AliCloud server is equipped with 4 quad-core 2.5 GHz Intel Xeon E5-2682 v4 and 16 GB RAM as the cloud server.

In addition, we build a base station based on open air interface. The base station works on Band7 (uplink 2500 MHz-2570 MHz, downlink 2620 MHz-2690 MHz). Note that, the base station is next to the edge server. The base station is responsible for communication and the edge server is responsible for providing computing and storage resources for the mobile application. The upload link rate of the Huawei honor 8 smartphone connected to the base station through LTE is 1, 000 KB/s, and the downlink rate is 1.36 MB/s. The upload link and downlink rate between the smartphone and the edge server via WiFi are 9 MB/s.

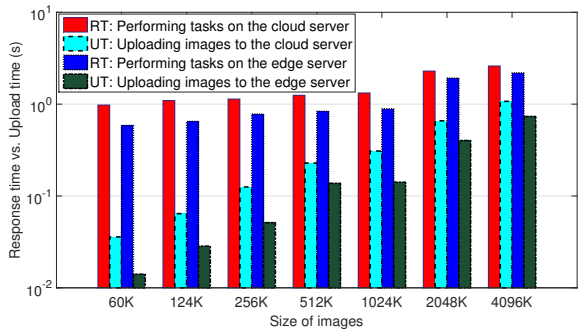


Fig. 6. Response time vs upload time of two different schemes in WiFi situation. RT indicates response time, and UT indicates upload time.

4.2.2 Response Time

EdgeCNN1-3 has a faster response. Fig. 4 illustrates the response time and upload time of two different schemes (*i.e.,* using CloudCNNr and EdgeCNN1-3 to provide cognitive services). Specifically, we observe that compared with CloudCNN, EdgeCNN1-3 can reduce the average response time by 55.08% (when the image data size is 60 K). This is because the edge server is closer to users and the structure of EdgeCNN1-3 is simpler.

Uploading images to the edge server can significantly reduce upload time. Fig. 4 shows that uploading images to the edge server can reduce the average upload time by 55.42% (when the image data size is 512 K) compared with uploading images to the cloud server. The reason is that users are closer to the edge server. In general, the distance between mobile devices and edge servers is one hop. Results illustrate the necessity of using edge servers to provide cognitive services.

EdgeCNN1-3 has shorter inference time. As illustrated in Fig. 5. Compared with CloudCNN, EdgeCNN1-3 can reduce the average inference time by 37.68% (when the image data size is 60 K). Because CloudCNN's structure is more complicated, it will consume longer inference time due to a large number of multiplication operations. This also shows that it is necessary to deploy a shallow model to meet the requirements of users' fast-responsive cognitive services.

In addition, we also investigate the response time and upload time in WiFi situation, as shown in Fig. 6. Fig. 6 shows similar observations in the case of LTE. For example, compared with CloudCNN, EdgeCNN1-3 can reduce the average response time by 40.26%.

In the 5G, the transmission rate between users and base stations is faster. Therefore, edge servers can provide faster response cognitive services.

4.3 Performance of the EdgeCNN

4.3.1 Experimental Setup

CloudCNN consists of four types of convolutional layers, one max-pooling layer, and one fully-connected (fc) layer with a final 10-way softmax. We only use one type of kernel with the size of 3×3, which is the smallest size to capture the notion of left/right, up/down, center [37]. The first type of convolutional layer consists of 3 layers with 32 kernels per layer. The second type of convolutional layer consists of 2 layers with 48 kernels per layer. The third type of convolutional layer consists of 5 layers with 80 kernels per layer. The fourth type convolutional layer consists of 5 layers with 128 kernels per layer. Fc layers have 1024 neurons.

EdgeCNN ends with a max-pooling layer and a fc layer with a final 10-way softmax. Similar, the fc layer has 1024 neurons. The max-pooling, dropout ratio and learning rate are same as CloudCNN. In CloudCNN and EdgeCNN, all convolutional layers are equipped with the ReLU [41].

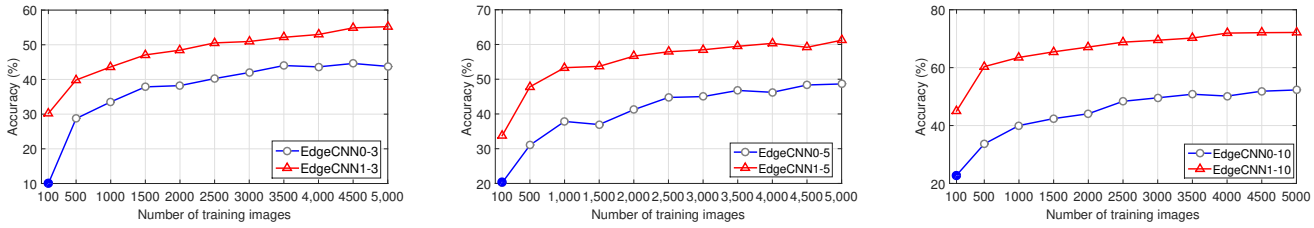
We first use the training set X_{trn} to train CloudCNN. EdgeCNN0 represents the shallow convolutional neural network model that is trained from scratch, and EdgeCNN1 represents the shallow CNN model that is trained with shared layers of CloudCNN. Their examples are shown in Fig. 3. Specifically, EdgeCNN0-3, EdgeCNN0-5, and EdgeCNN0-10 represent EdgeCNN0 with 3, 5, and 10 convolutional layers. EdgeCNN1-3, EdgeCNN1-5, and EdgeCNN1-10 represent EdgeCNN1 with 3, 5, and 10 convolutional layers. We build above models on TensorFlow.

The training procedures of CloudCNN and EdgeCNN follow [37], [41]. That is, the training is carried out by using mini-batch gradient descent with momentum. The batch size was set to 32, momentum to 0.9. Max-pooling is performed over a 2×2 pixel window, with stride 1. The dropout ratio is set to 0.25. We split the training set X_{trn} into a training set X_{trn1} and a validation set X_{val1}, and their ratio is 9:1. The learning rate was initially set to 0.01, and then decreased by a factor of 10 when the validation set X_{val1} accuracy stopped improving. In testing, we adopt the standard 10-crop testing. The accuracy of CloudCNN on MNIST, CIFAR-10, and FASHION-MNIST datasets are 99.48%, 87.80%, 93.50%, respectively. Training ends if the error did not drop for 30 epochs or if the learning rate was reduced by a factor of more than 1,000 in total. When the number of images is the least, EdgeCNN represents the edge model of the initialization phase. When the number of images increases, EdgeCNN represents the edge model of the update phase. For example, on the MNIST dataset, when the number of images is 120, EdgeCNN1-3 represents the edge model of the initialization phase. When the number of images is 240 or more, EdgeCNN1-3 represents the edge model of the update phase. To fair compare EdgeCNNs and CloudCNNs, we report the results on the test set.

4.3.2 EdgeCNN's Accuracy on CIFAR-10

EdgeCNN1 can significantly improve accuracy, especially when there is a small number of labeled images. Fig. 7 shows that the accuracy of six EdgeCNNs on CIFAR-10 dataset. Fig. 7 (c) shows that, compared with EdgeCNN0-10, the accuracy of EdgeCNN1-10 can be increased by 26.70% (when the number of labeled images is 500). The reason is that when the number of labeled images is small, EdgeCNN0 has difficulty converging and learning the distribution of the data when training from scratch. However, CloudCNN can easily learn the distribution of the data by using a considerable large number of labeled images on the cloud server. By inheriting the knowledge of the shared layers, the EdgeCNN1 can also easily learn the distribution of the data. Therefore, EdgeCNN1 achieves higher accuracy.

In addition, Figs. 7 (a), (b) and (c) show that when the number of images is 100, EdgeCNN0-3, EdgeCNN0-5 and EdgeCNN0-10 have low accuracy. This is because they suffer from over-fitting problems due to the small amount



(a) (b) (c)

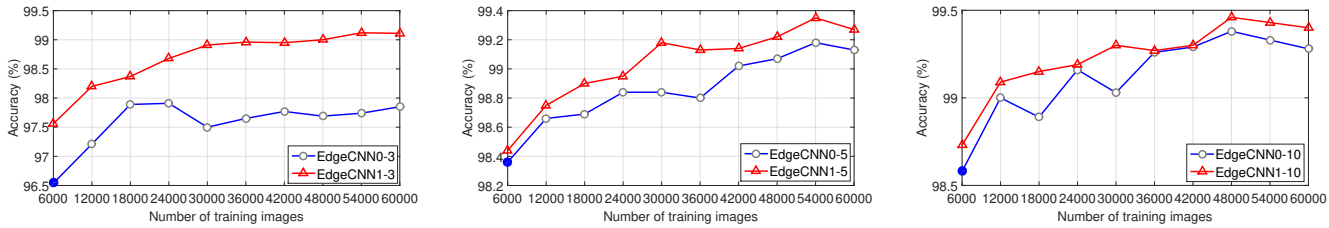
Fig. 7. Accuracy of EdgeCNN0 and EdgeCNN1 with different number of training images on CIFAR-10 dataset.

TABLE 2
Accuracy comparison of different models with different number of training images on MNIST dataset

Model/Number of training images	120	240	360	480	600	720	840	960	1080	1200
EdgeCNN0-3	74.36	82.96	88.35	89.04	90.00	90.45	90.68	91.98	93.67	93.46
EdgeCNN1-3	**83.12**	**88.97**	**89.76**	**90.17**	**91.97**	**94.04**	**94.65**	**95.31**	**95.41**	**95.47**
EdgeCNN0-5	83.38	91.94	91.12	93.75	93.40	93.61	95.08	95.35	96.40	96.54
EdgeCNN1-5	**84.39**	**92.81**	**93.49**	**94.14**	**95.50**	**96.09**	**96.40**	**96.56**	**96.65**	**96.81**
EdgeCNN0-10	84.75	89.91	93.01	94.69	95.04	95.85	95.31	96.11	94.62	96.19
EdgeCNN1-10	**85.17**	**94.51**	**95.14**	**95.12**	**96.20**	**96.42**	**96.49**	**96.99**	**97.05**	**97.20**

TABLE 3
Accuracy comparison of different models with different number of training images on FASION-MNIST dataset

Model/Number of training images	120	240	360	480	600	720	840	960	1080	1200
EdgeCNN0-3	58.61	68.74	72.01	74.40	75.93	75.98	76.06	77.08	78.01	78.77
EdgeCNN1-3	**75.77**	**78.67**	**82.04**	**83.39**	**84.05**	**84.34**	**84.46**	**85.49**	**85.59**	**86.00**
EdgeCNN0-5	63.91	69.34	77.38	78.23	78.73	78.54	79.17	79.69	80.73	80.37
EdgeCNN1-5	**78.21**	**82.55**	**84.15**	**84.53**	**86.12**	**86.26**	**86.63**	**87.66**	**87.80**	**88.23**
EdgeCNN0-10	65.66	71.47	76.02	76.14	77.84	78.23	78.55	80.36	81.00	81.22
EdgeCNN1-10	**77.59**	**82.61**	**84.15**	**85.68**	**86.55**	**86.48**	**86.78**	**87.64**	**88.01**	**88.23**



(a) (b) (c)

Fig. 8. Accuracy of EdgeCNN0 and EdgeCNN1 with different number of training images on MNIST dataset.

of training data. However, with the shared layers of the CloudCNN, EdgeCNN1-3, EdgeCNN1-5 and EdgeCNN1-10 have higher accuracy. The reason is that with the shared layers of the CloudCNN, these models can use the knowledge of the deep model to learn the distribution of data, rather than learning from scratch. This indicates that with the assistance of the deep model, EdgeCNN can avoid the over-fitting problem.

Continuously uploaded data can further improve the accuracy of EdgeCNN. Fig. 7 (a) shows that when the number of training data is increased from 100 to 5,000, the accuracy of EdgeCNN1-3 is increased by 25.03%. This indicates that with the labeled data provided by the CloudCNN, EdgeCNN can further improve accuracy. Because EdgeCNN0 does not have labeled data, it can only have initial accuracy. For example, Fig. 7 (a) illustrates the accuracy of EdgeCNN0-3 is 10% when the training data is 100. To compare EdgeCNN0 and EdgeCNN1 fairly, Fig. 7 also shows the accuracy of EdgeCNN0 as the labeled data increases. Fig. 7 (a) shows that the accuracy of EdgeCNN0-3 is increased by 33.77% when the number of training data is increased from 100 to 5000. This is because a large amount of training image data helps to characterize the distribution of image data. The results also show that for real-world deep learning model-based applications, it is crucial to collect sufficient trainable data.

4.3.3 EdgeCNN's accuracy on MNIST& FASHION-MNIST

We observe similar observations from the results of CIFAR-10 dataset that *with the shared layers of CloudCNN, EdgeCNN1 can improve its accuracy, and can further improve its accuracy with labeled data.* Table 2 shows that when the number of labeled images is 120, the accuracy of EdgeCNN1-3 is increased by 8.76% compared with EdgeCNN0-3. Table 3 shows that, compared with EdgeCNN0-3, when the number

(a) (b) (c)

Fig. 9. Accuracy of EdgeCNN0 and EdgeCNN1 with different number of training images on FASHION-MNIST dataset.

of labeled images is 120, the accuracy of EdgeCNN1-3 is increased by 17.16%.

In addition, we investigate the accuracy of EdgeCNNs when there is a large number of training images on MNIST and FASHION-MNIST datasets. As shown in Fig. 8 and Fig. 9. We observe that:

With the assistance of CloudCNN, the accuracy of EdgeCNN can be further improved even if the number of training images is large. On MNIST and FASHION-MNIST datasets, when the number of training images is 60,000, both CloudCNN and EdgeCNN are trained based on the same training set. CloudCNN can further improve the accuracy of EdgeCNN by sharing its m (3, 5, and 10) lower layers. Fig. 8 (a) shows that when the number of labeled images is 60,000, the accuracy of EdgeCNN1-3 is increased by 1.26% compared with EdgeCNN0-3. This may be because CloudCNN extracts better features (features that are beneficial to accuracy), and with the shared layers, EdgeCNN can move closer to these better features.

With the assistance of CloudCNN, the accuracy of EdgeCNN is close to the accuracy of CloudCNN with the same training set. When the number of training images is 60,000, Fig. 9 shows that the accuracy of EdgeCNN1-3, EdgeCNN1-5 and EdgeCNN1-10 is 93.15%, 93.30% and 93.09%. The accuracy of CloudCNN is 93.50%. This indicates the importance of using CloudCNN to assist in training EdgeCNN.

4.3.4 Training Time on Above Three Datasets

EdgeCNN1 has less training time. Fig. 10 shows the training time of different EdgeCNNs on three datasets when the number of training samples is 120, 100, and 120. To be exact, Fig. 10 shows that on CIFAR-10 dataset, EdgeCNN1-10 has 86.57% less training time than EdgeCNN0-10 layers; on MNIST dataset, EdgeCNN1-10 has 45.15% less training time than EdgeCNN0-10; on FASHION-MNIST dataset, EdgeCNN1-3 has 45.47% less training time than EdgeCNN0-3. This is because CloudCNN shares its m lower layers with EdgeCNN1. EdgeCNN1 inherits the knowledge of CloudCNN, it only needs to fine-tune the n higher layers, so it can have faster convergence speed. Therefore, EdgeCNN1 training with the shared layers of CloudCNN converges faster.

4.3.5 Inference Time on Above Three Datasets

The more layers are, the longer the inferring time is. Fig. 11 shows the inference time of different layers of EdgeCNNs on the CIFAR-10 dataset. As shown, when the number of test images is 2,000, the inference time of EdgeCNN1-3 is 51.87% and 23.28% less than the inference time of EdgeCNN1-10 and EdgeCNN1-5, respectively. The reason is

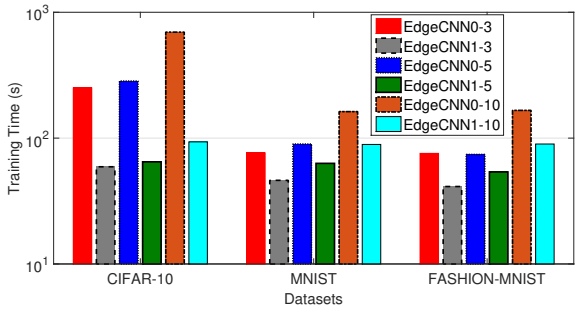


Fig. 10. Training time of different EdgeCNNs.

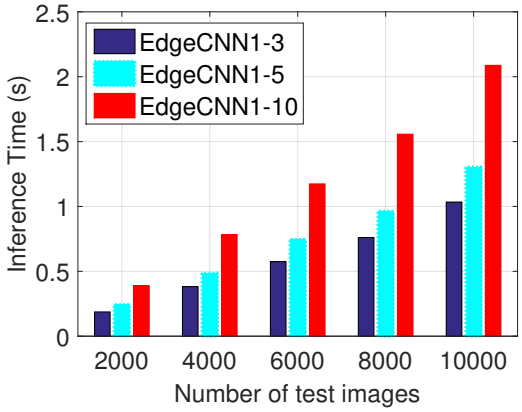


Fig. 11. Inference time of different EdgeCNNs.

the EdgeCNN1-3 has fewer layers. Multiple layers of deep learning models lead to longer inference time. Therefore, EdgeCNN1-3 has a shorter inference time.

4.4 Summary

In summary, the above results indicate that a faster response cognitive service can be provided by EdgeCNN because it is simpler and the edge server is closer to the user. In addition, EdgeCNN can achieve higher accuracy with the assistance of CloudCNN and continuously updated data. Hence, the proposed framework makes it possible to provide cognitive services with long duration, fast response and high accuracy.

It is worth noting that the proposed Cloud-Edge collaboration framework is applicable to general-purpose deep learning models. Existing high-performance neural networks can be elegantly incorporated. For instance, high-performance complex neural network models can be deployed on cloud servers as CloudCNN and we can design some shallow deep learning models to be deployed on edge servers as EdgeCNN. Through the Cloud-Edge collaboration framework proposed in this paper, EdgeCNN provides users with high-performance cognitive services. Experimental results show that with the assistance of the deep model,

the shallow model can achieve accuracy comparable to the deep model. This means that EdgeCNN can obtain the similar performance as existing high-performance models, such as ResNet.

5 Conclusion

In this paper, we propose a cloud-edge collaboration framework for cognitive services. The proposed framework is general-purpose for deep learning models and includes initialization phase and update phase. The initialization phase is static, using m lower layers of the deep model to assist in training the shallow model. The update phase is dynamic. The EdgeCNN is adaptive by using the uploaded data and the ongoing assistance of the deep model. Experimental results demonstrate the effectiveness of the proposed framework in terms of faster responses and higher accuracy.

In our future work, we plan to deploy different types of EdgeCNN on edge servers, and study the assistance mechanisms of CloudCNN for these different types of EdgeCNNs (*e.g.*, with different width and depth) to make them more accurate. That is, when each edge server receives the CloudCNN parameters sent by the cloud server, we plan to use scaling factors to expand or compress the received parameters to match the structure of each EdgeCNN. In addition, retraining EdgeCNNs on the edge server will consume the computing resources of the edge server. When there are many applications running on an edge server at the same time, it will cause computing resource competition. Therefore, studying when to retrain and rational use of edge server resources is also the focus of the research. In response to this problem, we plan to study the relationship between the accuracy of EdgeCNN and the number of training samples to determine the frequency of its retraining.

References

[1] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. B. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proceedings of the IEEE Symposium on Computers and Communications*. IEEE, 2012, pp. 59–66.

[2] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, Y. Ma, S. Chen, and P. Hou, "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Transactions on Service Computing*, vol. 11, no. 2, pp. 249–261, Mar. 2018.

[3] W. Wang, P. Xu, L. T. Yang, and J. Chen, "Cloud-assisted key distribution in batch for secure real-time mobile services," *IEEE Transactions on Services Computing*, vol. 11, no. 5, pp. 850–863, Sep. 2018.

[4] K.-C. Wu, W.-Y. Liu, and S.-Y. Wu, "Dynamic deployment and cost-sensitive provisioning for elastic mobile cloud services," *IEEE Transactions on Mobile Computing*, vol. 17, no. 6, pp. 1326–1338, Jun. 2018.

[5] H. Yin, X. Zhang, H. H. Liu, Y. Luo, C. Tian, S. Zhao, and F. Li, "Edge provisioning with flexible server placement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1031–1045, Apr. 2017.

[6] Y. Sarikaya, H. Inaltekin, T. Alpcan, and J. S. Evans, "Stability and dynamic control of underlay mobile edge networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 9, pp. 2195–2208, Sep. 2018.

[7] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, Mar. 2016.

[8] N. Fernando, S. W. Loke, and W. Rahayu, "Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 329–343, Jun. 2019.

[9] S. Wang, C. Ding, N. Zhang, X. Liu, A. Zhou, J. Cao, and X. Shen, "A cloud-guided feature extraction approach for image retrieval in mobile edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–14, Sep. 2019.

[10] J. Wu, M. Dong, K. Ota, J. Li, W. Yang, and M. Wang, "Fog-computing-enabled cognitive network function virtualization for an information-centric future internet," *IEEE Communications Magazine*, vol. 57, no. 7, pp. 48–54, Jul. 2019.

[11] H. Li, K. Ota, and M. Dong, "Deep reinforcement scheduling for mobile crowdsensing in fog computing," *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–18, Apr. 2019.

[12] C. Zhao, M. Dong, K. Ota, J. Li, and J. Wu, "Edge-mapreduce-based intelligent information-centric iov: Cognitive route planning," *IEEE Access*, vol. 7, pp. 50 549–50 560, Apr. 2019.

[13] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proceedings of the 37th International Conference on Distributed Computing Systems*. IEEE, 2017, pp. 276–286.

[14] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan. 2018.

[15] G. N. Karystinos and D. A. Pados, "On overfitting, generalization, and randomly expanded training sets," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1050–1057, Sep. 2000.

[16] Z. L, S. G, A. C, F. T, and H. B, "Nonsomatotopic organization of the higher motor centers in octopus," *Current Biology*, vol. 19, no. 19, pp. 1632–1636, Sep. 2009.

[17] X. Chen, L. Jiao, W. Li, and X. Fu, "Control of octopus arm extension by a peripheral motor program," *Science*, vol. 293, no. 5536, pp. 1845–1848, Sep. 2001.

[18] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 587–601.

[19] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *Proceedings of 25th USENIX Security Symposium*. USENIX Association, 2016, pp. 601–618.

[20] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proceedings of the International Conference on Neural Information Processing Systems*. MIT Press, 2014, pp. 3320–3328.

[21] Q. Hou, M.-M. Cheng, X. Hu, A. Borji, Z. Tu, and P. H. S. Torr, "Deeply supervised salient object detection with short connections," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 4, pp. 815–828, Apr. 2019.

[22] J. Han, R. Quan, D. Zhang, and F. Nie, "Robust object co-segmentation using background prior," *IEEE Transactions on Image Processing*, vol. 27, no. 4, pp. 1639–1651, Apr. 2018.

[23] A. P. Sheth, H. Y. Yip, A. lyengar, and P. Tepper, "Cognitive services and intelligent chatbots: Current perspectives and special issue introduction," *IEEE Internet Computing*, vol. 23, no. 2, pp. 6–12, 2019.

[24] A. lyengar, "Supporting data analytics applications which utilize cognitive services," in *Proceedings of the IEEE Conference on Distributed Computing Systems*. IEEE, 2017, pp. 1856–1864.

[25] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. N. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the 32th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 615–629.

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," in *arXiv:1704.04861*, 2017, pp. 1–9.

[27] Z. Fang, J.-H. Lin, M. B. Srivastava, and R. K. Gupta, "Multi-tenant mobile offloading systems for real-time computer vision applications," in *Proceedings of the 20th International Conference on Distributed Computing and Networking*. IEEE, 2019, pp. 21–30.

[28] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in

Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. ACM, 2016, pp. 123–136.
[29] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2018, pp. 6848–6856.
[30] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *arXiv:1503.02531*, 2015, pp. 576–584.
[31] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 2018, pp. 115–127.
[32] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proceedings of the 15th Annual International Conference on Mobile Systems*. ACM, 2017, pp. 68–81.
[33] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, "Fog computing based face identification and resolution scheme in internet of things," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1910–1920, Aug. 2017.
[34] A. Mehrabi, M. Siekkinen, and A. Y-Jääski, "Edge computing assisted adaptive mobile video streaming," *IEEE Transactions on Mobile Computing*, vol. 18, no. 4, pp. 787–800, Apr. 2019.
[35] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2017, pp. 843–852.
[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2016, pp. 770–778.
[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the 3th Internationa Conference on Learning Representations*. IEEE, 2015, pp. 1–14.
[38] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical report*, Apr. 2009.
[39] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 787–800, Nov. 1998.
[40] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," in *arXiv:1708.07747*, 2017, pp. 1–6.
[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the International Conference on Neural Information Processing Systems*. MIT Press, 2012, pp. 1106–1114.

Chuntao Ding received the B.S and M.S degrees from Sias University in 2012 and Soochow University in 2015, respectively. He is currently a Ph.D candidate at the State key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include Mobile Edge Computing, Deep Learning, Multi-task Learning.

Ao Zhou received the Ph.D degrees in Beijing University of Posts and Telecommunications, Beijing, China, in 2015. She is currently an Associate Professor with State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has published 20+ research papers. She played a key role at many international conferences. Her research interests include Cloud Computing and Edge Computing.

Yunxin Liu is currently a Senior Researcher and the research manager of Heterogeneous and Extreme Computing (HEX) group at Microsoft Research Asia (MSRA). He received his Ph.D. in computer science from Shanghai Jiao Tong University in 2011 (through the SJTU-MSRA joint PhD program), M.S. degree in computer science from Tsinghua University in 2001, and B.S. degrees in mechanical engineering and computer science from University of Science and Technology of China in 1988. He has broad research interests on mobile computing and edge computing, including power management, security and privacy, sensing, and intelligent edge/mobile systems. He is a senior member of the IEEE and a member of the ACM.

Rong Chang received his PhD degree in computer science and engineering from the University of Michigan in 1990. He is with IBM Research leading a global team creating innovative IoT cloud services technologies. He holds 30+ patents and has published 40+ papers. He is Member of IBM Academy of Technology, ACM Distinguished Engineer, Chair of IEEE Computer Society Technical Committee on Services Computing, Editor-in-Chief of the International Journal of Cloud Computing and Associate Editor-in-Chief of the IEEE Transactions on Services Computing.

Ching-Hsien Hsu is Chair Professor and Dean of the College of Information and Electrical Engineering, Asia University, Taiwan; His research includes high performance computing, cloud computing, parallel and distributed systems, big data analytics, ubiquitous/pervasive computing and intelligence. He has published 200 papers in top journals such as IEEE TPDS, IEEE TSC, ACM TOMM, IEEE TCC, IEEE TETC, IEEE System, IEEE Network, top conference proceedings, and book chapters in these areas. Dr. Hsu is a Fellow of the IET (IEE); Vice Chair of IEEE Technical Committee on Cloud Computing (TCCLD), IEEE Technical Committee on Scalable Computing (TCSC), a Senior member of IEEE.

Shangguang Wang received his Ph.D degree at Beijing University of Posts and Telecommunications in 2011. He is Professor and Deputy Director at the State Key Laboratory of Networking and Switching Technology (BUPT). He has published more than 150 papers, and played a key role at many international conferences, such as general chair and PC chair. His research interests include service computing, cloud computing, and mobile edge computing. He is a senior member of the IEEE, and the Editor-in-Chief of the International Journal of Web Science.