Optimization of Deep Learning Applications in Highly Heterogeneous Distributed System

Zongpu Zhang
Stu. ID: 515030910586
Contact: zhangz-z-p@sjtu.edu.cn
SJTU EE447 Course Project Report

May 25, 2018

Abstract

Existing distributed framework does not tailor for the fast development of Deep Neural Network (DNN), which is the key technique behind many intelligent applications nowadays. Based on prior exploration on Distributed Deep Neural Networks (DDNN) [1], a more generalized framework for AI applications over the distributed hierarchy is proposed. While being able to support basic functionalities of DNNs, this framework is optimized for various types of heterogeneity, including heterogeneous computing nodes, heterogeneous neural networks, and heterogeneous system tasks. Besides, this framework features parallel computing, privacy protection and robustness, with other consideration for the combination of heterogeneous distributed system and DNN. Extensive experiments demonstrate that this framework is capable of utilizing hierarchical distributed system better for DNN and tailoring DNN for real-world distributed system properly, which is with low response time, high performance, and better user experience.

1 Introduction

In recent years, thanks to the development of Deep Neural Network (DNN), many intelligent applications have changed people's life unconsciously, e.g., voice assistant and biometric authentication. Many algorithms in different fields have been revolutionized by algorithms with DNNs, for example, digital image processing [2], image classification [3], speech translation [4], speech recognition [5] and so on. However, it is observed that the requirement of computational resources involved in both training and testing phase of DNN is considerably high, which is traditionally solved by running on a powerful machine, or on a cloud.

On the field of distributed computing system, hierarchical scalable distributed computing system has made a fast progress recently [1, 6]. However, in the existing distributed computing frameworks, the calculation of DNN tasks is still treated the same as the other traditional intensive computing tasks. On the other hand, with the development of Internet technologies, computing nodes located in different places geographically are able to form a cooperative system providing ubiquitous computing services for users. Large number of end devices connected together, forming an Internet of Things (IoT), has become another hot topic in IT industry recently. With advantages like being located much closer to data provider, e.g., sensors, distributed system utilizing IoT devices is expected to be the next breakthrough of ubiquitous computing for the daily applications. As a type of well-organized framework, distributed systems are generally expected to have other features different from DNNs, e.g., high availability, high scalability and robustness over large scale and heterogeneous devices.

However, current DNNs only focus on performance improvement and single device deployment, such MobileNet [7] and SqueezeNet [8]. The utilization of distributed computing system is with less attention. A novel Distributed Deep Neural Networks (DDNN) framework [1] is proposed, trying to utilize the distributed system hierarchy and solve the above issue to some extend. However, it has certain limitations, including: (a). they more focus on the power eating training procedure on end devices, (b). they use

NN with exactly the same structure on all kinds of devices, and (c). they only validate their framework with limited task-specific experiments. To this end, a novel framework is proposed in this paper. On the one hand, it improves distribution computing framework in order to better support DNNs, including (a). distributed computing node heterogeneity for better resource utilization, (b). fully supported parallel computing, and (c). privacy protection and robustness. On the other hand, it also fits normal DNNs to the real-world distributed computing systems.

In this report, a novel distributed computing framework for AI applications over the cloud, the edge (fog) and the end devices (i.e., Internet of Things) is proposed. Intensive evaluations demonstrate the advantages of proposed framework, including short response time, high accuracy, better hardware usage, high scalability, privacy protection, and fault tolerance.

2 Related Works

In this section, recent researches related to distributed computing hierarchy (section 2.1), the application of deep neural networks to computer vision problems (section 2.2), and the framework of distributed deep neural networks (section 2.3) are reviewed.

2.1 Distributed Computing Hierarchy

Cloud computing has long been one of the most popular research directions in the IT industry since 2006 [9]. Its scalable infrastructures enable companies to run business on various cloud computing models, for instance, Google File System [10], and Apache Spark [11], and so on. With other functionalities offered by clouding computing, e.g. resource virtualization [12], software as a service (SaaS) [13], it has been widely used in our daily life. However, many problems are also observed in applications which utilize cloud computing, which may weaken the user experience, like long latency, data privacy, and so on [14,15].

Driven by both the development of 5G communications and network technologies, a paradigm shift in migrating cloud computation to network edges has been seen [16]. With the trend that more and more data are generated at the network edge, edge computing has been proposed hoping to process those data directly at edge devices [17]. Due to ongoing differences on the concept of edge and fog, edge is used mainly in the following discussion and do not explicitly distinguish them from each other. Previous works [18,19] have been introduced to enable such functionality, which can eliminate the drawback of long propagation delays compared with Cloud computing in real-world applications. Many benefits are provided by edge computing, like lower response time [20] and energy consumption [21], with several typical types of end devices, like gateway, micro data center, and cloudlet [19].

With the number of smart devices deployed around us increases rapidly, the next wave in the era of computing is predicted to be the Internet of Things (IoT) [14, 22, 23]. With the development of Internet technologies, sensor manufacturing, and Near Field Communication technologies, a significant boost in the number of sensors is observed in recent years [24, 25]. One of the consequences of such phenomenon lies in the fact that these sensors generate huge amount of data, which can not be handled by Cloud computing alone. With the sensor network, which is a key component of IoT, data can be quickly processed first on local end devices [22]. The connection of IoT devices can support many applications, including Healthcare system [26–28], Smart city / Smart home [29–31], Video surveillance [32, 33] and so on.

As the complexity of real-world applications and systems grows tremendously, a joint ubiquitous computing model is required by the industry. With that said, three layers of computational models, i.e. Cloud, Edge, and End, forms a new hierarchical structure. Such hierarchical structures are developed to meet various requirements, e.g., fast response, low latency, high availability, reliability, manageability, and low cost [6].

2.2 DNN for Computer Vision

More and more difficult problems have been significantly explored With the development of DNNs. As one of the most popular subjects in computer science society, computer vision problems have been treated as the test bed for various DNNs for a long time. Typical computer vision problems, such as object detection, object segmentation, image classification and video object tracking, have been well studied in recent years.

Many types of DNN architecture have been proposed to meet different real-world requirements, such as (a). high performance, (b). high speed, and (c). low utilization of both computational and memory resources.

A milestone for the application of neural networks in computer vision problems was the introduction of AlexNet [34], which considerably outperformed traditional methods in the ImageNet Large Scale Visual Recognition Challenge [35] in 2012. Later, convolutional neural networks which utilize smaller convolutional filters and deeper network architecture, e.g., VGG Net [36], was proposed for better performance on image classification. Residual Network [3] and Wide Residual Network (WRN) [37] are proposed to ease the difficulty of training a DNN by introducing residual functions which can efficiently pass the gradient back to shallow layers in the back-propagation phase. Besides, Binarized neural network (BNN) [38], MobileNet [7] are proposed as lightweight neural networks for mobile and embedded devices, with the concept of streamlined architecture. Extensive experiments shows that they can achieve acceptable performance overhead with much less computation and smaller model size compared with state-of-the-art structures.

2.3 Distributed DNN

Related work of applying large-scale DNNs on distributed systems traditionally focuses on speeding up the training phase. Some frameworks have proved the efficiency of distributing neural networks on large-scale computer clusters with the help of both data and model parallelism. DistBelief framework [40] employs an asynchronous stochastic gradient descent procedure, Downpour SGD and a distributed batch optimization framework, Sandblaster. On the other hand, FireCaffe [41] utilizes reduction tree based synchronous stochastic gradient descent algorithm and at the same time implements the framework on a large scale graphics processing unit (GPU) cluster. Different optimization algorithms, specifically different SGD algorithms (e.g., Gossiping SGD [42]) have also been proposed for better performance and lower training time of DNNs on distributed systems.

However, much less research has been done for distributing DNNs on a hierarchical distributed system, where the hardware, location and network resources are dramatically different for each type of computing nodes. In [1], a distributed framework was proposed to apply DNNs on the cloud, the edge (fog) and end devices, where the DNNs are patitioned and the shallow part is put on less powerful computing nodes (edge/end devices) with an algorithm determining the local exit point. The system can scale both horizontally (for neural network size) and vertically (for geographical span) with features, such as fault tolerance and privacy protection.

3 The Proposed Framework

As illustrated in Figure 1, the proposed framework consists of three levels of heterogeneity: (a). distributed computing node heterogeneity, (b). computing node neural network heterogeneity, and (c). system task heterogeneity. Based on the observation of real-world situations and the research on related works, the following assumptions are made during the discussion:

1. Only focus on the testing phase of neural network algorithms, rather than the training phase.
2. All tasks are assigned from end devices, in a bottom-to-up way. For simplicity, only the case where one end device assigns tasks is considered.
3. End devices cannot process for a long time due to power limit. Edge devices are more powerful than end devices, and power limit is ignored on edge devices. Cloud does not have limitation of computational resources and power consumption.
4. For privacy protection, all end devices are treated as dependable, some edge devices are treated as dependable, and the cloud is regarded as undependable.

End devices stand for the devices that are relatively close to the end user. For example, mobile phones owned by users, IoT devices inside a smart home, and smart wearable devices on the end user are typical end devices. Since end users suffer from low accuracy, low throughput, and high device occupancy, it is not a preferred methodology to simply process data at end devices. The cloud represents devices that are

far away from the end user, which is generally more powerful than other devices. Devices stay between end devices and the cloud are called edge devices, such as routers and small data centers located between users and the service provider.

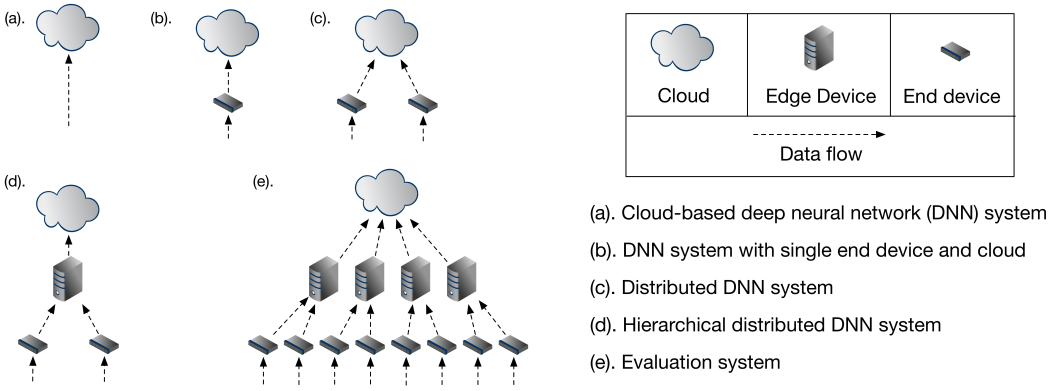

Figure 1: Illustration of distributed deep neural network system over the cloud, edge devices and end devices

3.1 Distributed Computing Node Heterogeneity

The proposed framework is built upon hierarchical distributed systems, including cloud, edge devices, and end devices. Existing ways for deep learning based service providers to handle data from end users involves uploading all data to their processing node, e.g. a cloud. However, such framework has many disadvantages,

1. Long response delays are likely to be generated, due to unpredictable network connection quality between end users and the service provider.

2. Large amount of data are generated every second with the development of IoT devices, which makes conventional computing unable to handle the task.

3. The computational resources on other end devices around the end user, and edge devices between the end user and the cloud are wasted.

4. The privacy issue is likely to happen when uploading sensitive data to the cloud.

Instead of relying on a cloud remotely, this framework distributes the computing on different types of computing nodes. Since different types of nodes have their own features, this framework tries to meet their requirements and makes the best use of them. Besides better hardware usage, distributing computing node heterogeneity also requires considering the network latency for different types of devices. Based on the assumption that all tasks are generated from end devices, the hierarchy of end devices, edge devices, and the cloud naturally represents the network latency for them. Other end devices located in the same network, or close to the task assigner have the lowest network latency. Edge devices then have moderate network latency, and the cloud has the highest latency. By setting tasks for end devices and edge devices, proposed framework can be adaptable to different network configurations and efficiently process the data.

3.2 Neural Network Heterogeneity

Instead of loading neural networks with the same structure on all types of devices in the distributed system hierarchy [1], devices in the proposed framework are loaded with different neural networks. It is straightforward that loading a binary network on the cloud is a waste of computing resources, and on the other hand, loading a full-size DNN on end devices is unrealistic. The goal of enabling neural network

heterogeneity is to fully utilize the features of each device, including memory limitation, computational capability, and network status, etc.

For the three types of devices, different factors should be emphasized while choosing the neural networks for it. The cloud has the most powerful hardware in the distributed hierarchy. Therefore, the primary metric for choosing neural networks on the cloud is the performance, e.g., the lowest error rate for image classification. Edge devices have the medium computational capability meanwhile they are significantly closer to end users. DNNs which balance between the performance and time delay (including processing time on edge devices, and communication overhead) are chosen for them. End devices have the most restricted condition, where choices of neural networks could be limited by memory size and computational capability. A shallow or more optimized neural network should be chosen for end devices. Factors including less response time, low power consumption, and low memory usage should be considered ahead of the performance of neural networks.

3.3 System Task Heterogeneity

Heterogeneous system task are proposed for the consideration of real-world situations. Sometimes a coarse-grained result is enough for the user to draw a conclusion, or react correspondingly, e.g. knowing certain object as one kind of 'fruit' is sometimes enough instead of identifying the exact category, e.g., 'apple'. With system task heterogeneity, neural networks that are loaded on devices with fewer computational resources can be modified to output a coarse-grained response. By doing so, instead of forcing the neural network to output a more meaningful but computationally expensive fine-grained response, the calculation time of such devices can be improved to a usable level.

This framework is designed to load different neural networks for different tasks on different types of machines in the distributed hierarchy. Relatively shallow networks for coarse-grained tasks are loaded on end devices for fast response time, low memory requirement and low power consumption. DNNs for fine-grained tasks are loaded on edge devices, which produce much better results than the end devices. Even stronger DNNs are loaded on the cloud for the highest possible performance of fine-grained tasks. Resources on each type of computing nodes are fully utilized and optimized, and the overall system outperforms the individual device in different ways, such as the response time and the accuracy.

3.4 Scheduling Algorithm and Scalability

Scheduling algorithms are important for optimizing the behavior of the proposed system with data intensive jobs. To formulate and compare several types of scheduling, four schemes are presented representing four different types of real-world situations. Tasks to be processed are first sent to weaker nodes, then to stronger nodes for better results. For simplicity, suppose each task is processed at most on two types of devices in the distributed hierarchy, and the first type of device is end device. Suppose that the current job involves processing N_t tasks, and there are end devices (N_{end}), edge devices (N_{edge}), and clouds (N_{cloud}) in the distributed system. The communication latency for end device is denoted by t^c_{end}. The latency from end devices to edge devices and cloud are denoted by t^c_{edge}, and t^c_{cloud} respectively, all of which include both the sending latency and the receiving latency. The processing time for each task on each type of device is t^p_{end}, t^p_{edge}, and t^p_{cloud} respectively. In the following discussion, it is assumed that the speed of the job is only determined by the scheduling scheme, processing time, and communication time. Other factors, e.g., the speed of assigning tasks to other devices, is not taken into consideration.

As a baseline, scheme 'End' involves sending tasks to all end devices in a sequential order (e.g., tasks are sent to end devices #1, #2, ..., $N_{end}-1$, N_{end}, #1, #2, ...). In ideal cases, the time to complete this job is

$$T_{End} = \frac{N_t}{N_{end}} \times (t^c_{end} + t^p_{end}) \tag{1}$$

Since end devices are generally close to end users, the network latency for end devices is considerably low, which means the time to finish the job is mainly controlled by the number of end devices N_{end} and the processing time on end devices t^p_{end}. The 'End-Cloud' scheme is one extension of the 'End' scheme, where all tasks are first processed on end devices, then sent to the cloud for further processing. In ideal cases,

still with sequential order of assigning tasks for the cloud, the time to complete this job is

$$T_{End-Cloud} = T_{End} + \frac{N_t}{N_{cloud}} \times (t^c_{cloud} + t^p_{cloud}) \tag{2}$$

Since the network latency for cloud is generally large and unstable, the communication overhead then becomes the bottleneck of the 'End-Cloud' scheme in practice compared with the effect from T_{End}, t^p_{cloud} and N_{cloud}. The 'End-Edge' scheme is then a trade-off between processing time and performance, where in ideal cases

$$T_{End-Edge} = T_{End} + \frac{N_t}{N_{edge}} \times (t^c_{edge} + t^p_{edge}) \tag{3}$$

With moderate network delay, communication overhead is no longer the bottleneck of the entire system. Also, since memory and computational resources on edge devices are considered much better than end devices, it is both affordable and effective to adopt relatively deeper neural networks on edge devices to balance the processing time and performance.

However, the above three schemes cannot fully utilize the distributed hierarchy because at most two types of devices are used while processing the job. Note that devices in a higher hierarchy (e.g., edge devices V.S. end devices) generally require more time for each task because of higher network delay and longer processing time due to deeper neural networks. Thus a mapping based scheduling method is proposed in the framework. The 'Mapping' scheme considers the ratio of total processing time between end devices and edge devices, and assign the upper-level processing unit for each end devices instead of assigning sequentially. Intuitively, the number of end devices mapped to one edge device is calculated by

$$N_{map} = \lceil \frac{t^c_{edge} + t^p_{edge}}{t^c_{end} + t^p_{end}} \rceil \tag{4}$$

which means that each subset of N_{map} end devices sends the workload to one edge device, so that the total time for N_{map} end devices to process the next batch of input data is no less than the time for one edge device to process. When all edge devices have been assigned to a certain number of end devices, the rest end devices are applied the same algorithm and mapped to the cloud. Tasks for end devices are still scheduled in sequential order for the highest usage of computational resources on end devices. The ratio of tasks sent to the cloud is calculated by

$$\theta_{cloud} = \frac{N_{end} - N_{edge} \times N_{map}}{N_{end}} \tag{5}$$

Ideally, the processing time for 'Mapping' scheme is

$$T_{Mapping} = \theta_{cloud} \times T_{End-Cloud} + \theta_{edge} \times T_{End-Edge} \tag{6}$$

where $\theta_{edge} = 1 - \theta_{cloud}$, which stays between $T_{End-Cloud}$ and $T_{End-Edge}$.

Two advantages of applying the 'Mapping' scheme are straightforward: (a). by offloading certain ratio of tasks to the cloud, the precision can be improved from the 'End-Edge' scheme with controllable processing time overhead from the 'End-Cloud' scheme, and (b). higher utilization of devices in the distributed hierarchy. Since the main problem for the 'End-Cloud' scheme in practice is the bottleneck of high network delay in the cloud side, the 'Mapping' scheme reduces such effect by assigning part of the job to cloud, and the rest to edge devices for lower response time. The scalability of the 'End' scheme and the 'End-Edge' scheme is theoretically better than the 'End-Cloud' due to the communication bottleneck of the 'End-Cloud'. However, it is observed in practice that other factors, e.g., the speed of assigning tasks, become the bottleneck of the 'End' scheme, because the total time for the 'End' is relatively small. The scalability of 'Mapping' is more complicated: it is closer to 'End-Edge' when the number of end devices cannot satisfy the requirement of edge devices, but it is closer to the 'End-Cloud' when the number of end devices are too large.

3.5 Privacy Protection

Privacy issue could happen during the communication with untrusted devices, or when the data is uploaded to a cloud. For example, hackers may intercept the transmitted data while being transferred to another

device through an open network, or attack the cloud service provider to steal users' private data. This framework prevents such cases from happening by encrypting sensitive data with a single layer neural network stripped from the neural network in a higher device hierarchy. The first convolutional layer of the neural network in the cloud (or edge) is downloaded on the end device as a light-weight encryption module. Instead of sending the source data, the output data from the encryption module, which is a multi-dimensional matrix containing floating point numbers, is sent to untrusted devices if necessary.

One of the concerns of using convolutional layers to encrypt the source data is the computational overhead. Given a specific task, suppose that the total processing time without encryption is t_{wo}, and the time with encryption is t_w. The overhead of processing time can be expressed as below.

$$Overhead = t_w - t_{wo} \tag{7}$$

With more detailed analysis, the overhead can be expressed mainly as the sum of two parts: (a). the difference of processing time of calculating the source data with those layers for encryption on two devices, and (b). the difference of communication time of transferring the encrypted data and source data, i.e.

$$Overhead = \Delta t_{encrypt} + \Delta t_{comm} \tag{8}$$

The former time difference can be reduced by offloading the calculation of encryption module on more powerful computing nodes (e.g., switch from end devices to edge devices), while the latter one can be reduced by reducing the output size of encryption module.

3.6 Fault Tolerance

To recover from device failure, two methods which enable fault tolerance of the system are considered. In the first method, after all tasks have been assigned at least once, unfinished tasks are reassigned to all devices that are available at the very beginning regardless of their current state. This method is relatively easy to implement, but it lacks flexibility in dealing with device failure. Communication time overhead is generated whenever a task is assigned to a failed device, which triggers the timeout limit of the communication module. Specifically, suppose that the system has sent N_t tasks to failed devices, and the communication timeout is set to t_{comm}, then the overhead of communication time is

$$Overhead = N_t \times t_{comm} \tag{9}$$

which is unacceptably large when many devices have failed.

To tackle the communication overhead, another method is considered which dynamically monitors the state of each device. Another process on the device which is responsible for sending tasks is started to query all available devices in a device list. Any device that does not respond to the query, or trigger a communication timeout is deleted from the device list. Only devices in the list can receive tasks, so that the number of communication timeout can be significantly reduced.

The interval of queries is important for end devices due to their limited computation and network resources. With a small query interval, the list of available devices is updated quickly so that communication timeout is less likely to happen. However, the computation involved in computing and communicating may slow down other parts of the end devices. Thus a balanced query interval should be chosen according to real-world situations.

4 System Evaluation

In this section, the evaluation of proposed framework are presented. Various advantages of this framework are available, including but not limited to the following aspects:

1. The proposed framework works on heterogeneous distributed computing systems and takes advantage of their own features for better performance, lower response time, and better resource usage.

2. The proposed framework loads different neural networks on different devices to balance the response time, performance, and resource usage.

3. The proposed framework takes advantage of heterogeneous computation tasks to efficiently offload easier tasks to devices with less computational ability for better user experience.

4. The proposed framework utilizes mapping based scheduling algorithm for balanced response time and performance.

5. The proposed framework achieves privacy protection by using a single-layer neural network for encryption and separating dependable and undependable devices.

6. The proposed framework features fault tolerance by monitoring the state of each device and maintaining a list of available devices.

First the structure of neural networks used in the evaluation system is presented in Section 4.1. Then the evaluation system architecture is presented in Section 4.2. Section 4.3 introduces the evaluation dataset, and Section 4.4 to 4.6 presents the impact of three types of heterogeneities. Section 4.7 to 4.9 presents the results regarding to scheduling algorithm and scalability, privacy protection, and fault tolerance.

4.1 Neural Networks In The Evaluation System

Instead of loading the same model on all types of devices [1], the proposed framework introduces an approach to load different neural networks on different types of devices to better accommodate their features and restrictions. For end devices, which has very limited memory and computational resources, a tiny but powerful neural network, MobileNet [7] is being used. Since the restriction of hardware is relatively loosen for edge devices, a much deeper, much stronger, and widely used DNN structure, Residual Network (ResNet) [3] is being used. The detailed structures of aforementioned Neural Networks are shown in Figure 2. The choice for cloud devices is all about performance, i.e., the accuracy of classification for the evaluation dataset. A variant of Residual Network, the Wide Residual Network [37], which is both larger in model size and deeper than typical ResNet configurations, is being used. All models are trained off-line with the entire CIFAR100 training set. Some details about the three types of models and the training settings are listed below.

The MobileNet network structure follows the original setup reported in the original paper, with two global hyper-parameters: width multiplier $\alpha = 0.5$, and depth multiplier $\beta = 1$. The input size of MobileNet is changed to the dimension of images in CIFAR100 dataset, which is $[32 \times 32 \times 3]$, and the output size is changed to 20 for exactly 20 classes of coarse labels. The model is trained with batch size of 32 for 100 epochs. The training data augmentation methods reported in the original paper is followed, and Adam [43] is being used as the optimizer with an initial learning rate of 0.01.

On edge devices, ResNet with a 34-layer configuration is being used. The input size of ResNet is changed to fit the training images, and the output layer can produce classification for 100 labels, which is the number of fine-grained labels. The ResNet is trained with batch size of 32 for up to 500 epochs, and the model with the best performance (i.e., the lowest loss) is saved as the final model. The training data augmentation includes feature-wise mean subtraction, squeezing and randomly horizontally flip. Adam is used as the optimizer.

Wide Residual Network (WRN) is being used for the best accuracy among all types of devices, which is trained for classification of 100 labels. It is configured with depth parameter of 28, and width parameter of 10 in the model. It is trained with Stochastic Gradient Descent with Momentum of 0.9 and Nesterov method enabled. The learning rate starts from 0.1, and decays by a factor of 0.2 when the number of epochs reaches 60, 120, and 160. The training data is augmented by feature-wise centering, standard deviation normalization, and ZCA whitening. The best model trained for 200 epochs, with batch size of 128 is chosen as the final model for testing.

4.2 Evaluation System Architecture

The evaluation system architecture is illustrated in Figure 1. (e), which contains 8 end devices, 4 edge devices, and 1 cloud. These virtual machines (VMs) are being used: 8 separate VMs each with one virtual

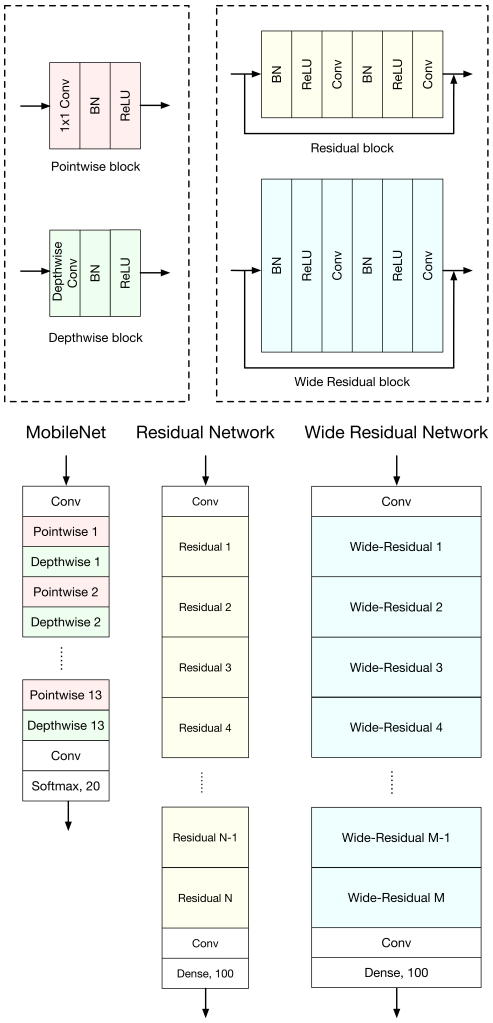

Figure 2: Network structure of MobileNet [7], Residual Network (ResNet) [3], and Wide Residual Network (WRN) [37].

CPU core running Ubuntu 16.04 as test machines for 8 end devices, 4 VMs each having 4 CPU cores for 4 edge devices and 1 terminal on the host machine with GPU acceleration enabled for the cloud.

The communication between devices are implemented by HTTP requests, through a virtual network interface. To simulate the real-world network delay between computing nodes, which cannot be implemented on hardware level in the evaluation system, I manually setup time delays for the communication between different types of devices. In the following evaluation, if not specifying, the simulated time delay for the cloud is 1 second, and 0.5 seconds for edge devices.

4.3 Evaluation Dataset

The system is evaluated on CIFAR100 [44], which is a challenging image classification dataset containing 60K 32x32 colored natural images in total. There are originally 100 fine-grained labels in this dataset, representing 100 types of objects appeared in all images (e.g. apples, crab, and fox). 20 classes are proposed as the abstraction of those 100 classes, which is called superclasses, or coarse-grained classes. Accuracy is chosen as the metric for evaluating the performance, which is the ratio of correct predictions among all test cases.

There are three main reasons for choosing this dataset in the evaluation system: (a). the image size is relatively small, which makes the simulation of computing on end devices possible, (b). the hierarchy

structure of labels in this dataset meets the requirement of system task heterogeneity, and (c). the dataset is widely used for testing image classification algorithms. In the following discussion, if it is not explicitly explained, the proposed framework is being tested for the classification job on the entire test set. The default batch size is 50, which means that each communication transfers 50 images sampled from the test set or 50 results correspondingly. Data pre-processing is implemented in the module of each neural network, and no global data pre-processing is done for test images. By default, the performance concerning accuracy or time through all tasks are averaged over 10K/50 = 200 runs.The accuracy of coarse labels and fine labels are reported separately.

4.4 Impact of Distributed Computing Node Heterogeneity

To illustrate the impact of distributed computing node heterogeneity, I setup a baseline where the theoretical inference time, the classification accuracy for fine-grained and coarse-grained labels and the size of models are reported in Table 1. The speed is tested by launching a simple program, which is without communication latency and continuously feeds the neural network with batched images. The difference of time-stamp when each task starts and when it is finished is treated as the theoretical inference time for such neural network on a typical type of device for a single batch of images. It is then averaged over 200 runs, as explained at the end of Section 4.3. Without distributed computing node heterogeneity, mainstream solutions involve loading a powerful neural network on the cloud. The computational resources on edge and end devices are then wasted because they only take charge of generating source data or transferring those data. By distributing neural networks on end and edge devices, they can produce classification result together with the cloud. For example, suppose that MobileNet is loaded on end devices, ResNet on edge devices, and WRN on the cloud. Theoretical inference time for the three networks are relatively close to each other, with up to only 34% differences for the comparison between Cloud+WRN and End device+MobileNet. It shows that instead of uploading all data to the cloud for computing, making use of the distributed hierarchy is fully functional.

Table 1: Accuracy, model size, and theoretical inference time for different neural networks on different types of computing nodes. The difference of the time-stamp when each task starts and when it is finished, divided by batch size (50 in practice) is referred to the inference time for each image (ms/image.). The accuracy for coarse and fine labels (20 and 100 labels) are reported separately.

Inference time (ms/image) ↘	MobileNet	ResNet	WRN
Cloud	0.25	0.79	6.99
Edge device	2.03	5.53	227
End device	5.20	15.7	1.17K
Accuracy - 20 labels	0.50	0.73	0.83
Accuracy - 100 labels	/	0.61	0.73
Number of parameters (M)	0.84	21.4	40.6
Size of weights file (MB)	10.3	85.7	324.9

Another factor that is important with respect to the impact of computing node heterogeneity, is the consideration of network latency. I test the speed of processing a batch of images on various devices with and without network latencies in the actual distributed system. The latency for edge devices and the cloud are set to 0.5 (s) and 1 (s) respectively throughout the following experiments. I present the inference time tested in the real system as well as the overall process time per image in Table 2. The process time is the sum of inference time for a batch of images and communication latency, measured in the simulation system, which represents the actual processing power of the system. Due to computational overhead from other parts of the program, such as HTTP server, data packing and unpacking, and process

synchronization, the actual inference time is slightly larger than theoretical inference time from Table 1. With computing node heterogeneity, the processing time has decreased a lot, from 31.0 (ms/image) to 10.4 or 18.3 (ms/image) for end and edge devices. The advantage mainly comes from avoiding the long network latency that is probably unavoidable in traditional framework with only the cloud.

Table 2: The speed of different neural networks on different types of devices, with various communication latency, in the actual distributed framework. The communication latency includes both sending latency and receiving latency. 'Inf. time' represents the inference time per image, which is time purely from neural network calculation. 'Proc. time' represents the overall processing time per image.

Type of devices	End dev.	Edge dev.		Cloud	
Neural network	MobileNet	ResNet		WRN	
Inf. time (ms/image)	7.41	5.99		7.36	
Comm. latency (s)	0	0	0.5	0	1
Proc. time (ms/image)	10.4	8.4	18.3	10.8	31.0

4.5 Impact of Neural Network Heterogeneity

One of the motivations for adopting neural network heterogeneity is to speed up the inference time on computing nodes with low computational resources, e.g.IoT devices. To demonstrate the impact of neural network heterogeneity, I also systematically compare the inference time, model size, and inference speed of each neural network on different types of devices, as illustrated in Table 1.

The model size is very important for end devices, especially with the rapidly growing depth of modern neural networks. Instead of loading giant neural networks on end devices, neural network heterogeneity allows us to put a compact neural networks with much fewer parameters and a much smaller weight file. As shown in the last two rows in Table 1, I managed to squeeze a neural network to an end device, which has up to 48x fewer parameters, and up to 32x smaller in terms of the size of weights file by comparing MobileNet with WRN. The benefit of smaller models on end devices is straightforward in terms of speed, where it achieve 3x and 225x less inference time compared with neural networks on edge devices and the cloud if they are tested on the same end device. On the other hand, if WRN is loaded on an end device, the inference time of 1.17(s) per image is going to block all other processes on the end device, making it totally unusable. The drawback in accuracy is less important for end devices, since one of the goals of distributing tasks on end devices is the low response time.

On the other hand, loading much deeper and stronger neural networks on edge devices or the cloud enables them to perform much better than end devices. Specifically, ResNet on edge devices is 46% better in terms of coarse label accuracy compared with MobileNet on end devices, and WRN on the cloud is 20% better regarding fine label accuracy compared with ResNet on edge devices. Since edge devices and the cloud do not have limitations on power consumption and moreover the cloud has no limited computational resources, they can deal with complex computation quickly. With more CPU cores enabled on edge devices (compared with end devices, where 4 vCPUs V.S. 1 vCPU in the simulation), edge devices can handle the inference of ResNet, which is 25x larger than MobileNet in terms of the number of parameters and with 2.8x less inference time than on end devices. The cloud, which is equipped with GPU produces even better speedup for larger and deeper neural networks. The inference time of WRN on edge devices is 32x more than that on the cloud, but the inference time of MobileNet on edge devices is only 8x more than that on the cloud, meaning that the cloud is even more suitable for a large scale DNN. In summary, with the help of neural network heterogeneity, all the three types of devices are better utilized according to their features, and the performance and response time for the system are enhanced.

4.6 Impact of System Task Heterogeneity

System task heterogeneity actually makes the neural networks on end devices realistic. It enhances user experience by generating a coarse response for less response time on a less powerful computing node. The accuracy of MobileNet for coarse labels is relatively usable, considering its fast inference speed. Combined with distributed computing node heterogeneity and neural network heterogeneity, the actual workflow of the evaluation system is demonstrated as below:

1. Test images are first distributed to end devices (or locally, in the perspective of task assigner) for a quick, coarse-grained response.

2. Certain data is encrypted to protect the privacy of users if sending to undependable computing nodes, which is discussed in Section 3.5.

3. According to the scheduling algorithm in Section 3.4, data is further sent to edge devices or the cloud for a slower, fine-grained response.

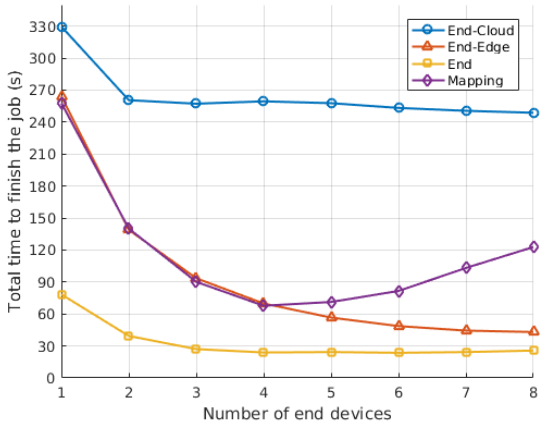

Figure 3: The time to finish the classification task in different situations with various numbers of end devices. In all 4 schemes, end devices first produce a coarse response given the source image. In the End-Cloud and End-Edge scheme, the data is sent to the cloud and edge devices for further process. In the End-Edge Mapping scheme, the first 4 end devices send data to 4 edge devices correspondingly, and the rest of end devices send data to the cloud.

4.7 Evaluation for Scheduling and Scalability

Based on the discussion in Section 3.4, the impact of various scheduling algorithms and the scalability of proposed framework is tested. Here I use the job of classifying the entire CIFAR100 test set, with batch size of 50 for each task, and compare the total completion time. The encryption module is enabled in this section for more realistic simulation. Four situations are considered: *End*, *End-Cloud*, *End-Edge* and *Mapping*. The curve of the duration for four schemes (T_{End}, $T_{End-Cloud}$, $T_{End-Edge}$, and $T_{Mapping}$) with respect to the number of end devices is illustrated in Figure 3.

It is observed that both *End-Cloud* and *End* situations have relatively poor scalability. In practice, as the number of end devices increases, the network delay between end devices and the cloud becomes the bottleneck for the *End-Cloud* scheme. For the *End* scheme, the speed of assigning tasks becomes the bottleneck as the number of end devices becomes larger. For the *End-Edge* and the *End-Edge Mapping* scheme, the usage of edge device improves the scalability of distributed systems. There are mainly three reasons contributing to such advantages: (a). accessible multiple edge devices, which increase the throughput compared with one cloud device in *End-Cloud* scheme, (b). relatively lower network delay (0.5(s) for edge devices V.S. 1(s) for the cloud), compared with *End-Cloud* scheme, and (c). relatively

higher inference speed than end devices, which makes the total processing time become closer and closer to the *End* scheme with more and more end devices. For the *End-Edge Mapping* scheme, N_{map} is set to 1 according to the evaluation system setup. The minor difference between the *Mapping* and the *End-Edge* scheme with 1-4 end devices in Figure 3 shows the efficiency of mapping end devices to edge devices according to their relative ratio of processing time. However, the communication time for the cloud restricts the scalability when more than 4 end devices are used in the *Mapping* scheme.

4.8 Additional Cost for Privacy Protection

I use a single layer of convolutional layer stripped from the deeper neural network in the cloud or the edge device as the encryption module in the evaluation system. Table 3 illustrates the output size, the number of parameters, and the size of weight file of the encryption module for two types of DNNs on edge devices and the cloud. For comparison, the number of parameters in encryption modules for ResNet and WRN is only 1.1% and 0.05% of parameters in MobileNet, showing that the computational overhead is subtle.

Table 3: The output dimension, number of parameters, and size of weights file of encryption module. Two types of encryption modules: (a). module for ResNet on edge devices, and (b). module for WRN on the cloud.

Encryption module for:	ResNet	WRN
Dimension of output	$16 \times 16 \times 64$	$32 \times 32 \times 16$
Number of parameters	9472	432
Size of weights file (KB)	50.1	11.9

In the evaluation system, the overhead is tested with and without encryption module, with different simulated values of network delay. Table 4 illustrates the overhead of processing time for each task (explained in Section 4.3) in two situations: (a). encrypt the data on an end device, then send to the cloud, and (b). encrypt the data on an edge device, then send to the cloud. It is observed that the difference of communication time Δt_{comm} is subtle compared with the difference of processing time for encryption. By encrypting on a stronger device, the overhead can be reduced from 13.1% (at 2(s) latency) to 29.8% (at 0.5(s) latency). Besides, the time overhead is less significant as the network latency goes higher (e.g., the overhead drops from 57.1% at 0.5(s) latency to 17.9% at 2(s) latency for end-cloud scheme). Since the network latency in real world is generally large and unstable, the experiment result suggests that the overhead in real system is considerably small. Further optimization on scheduling algorithm can be done for less overhead given the latency and computational capacity of each device.

Table 4: The time overhead with different simulation values of network latency. The overhead is the percentage of additional time used with encryption module, compared with no encryption module used. In both cases, the encrypted/unencrypted is finally sent to the cloud for calculation.

	Encrypt at	Network latency			
		0.5	1	1.5	2
Overhead	End dev.	57.1%	33.2%	24.7%	17.9%
	Edge dev.	44.0%	27.2%	20.6%	15.9%

4.9 Showcase of Fault Tolerance

Three methods with respect to the requirement of fault tolerance are tested in the evaluation system: (a). without any functionality dealing with device failure, (b). reassigning unfinished tasks sequentially

through all devices regardless of their state after all tasks have been assigned once, and (c). monitor the availability of related devices and maintain a list for all devices that are currently available. I test these methods with the job of classifying the entire test set of CIFAR100, with batch size of 50 for each sub-task. The duration from the beginning of assigning tasks to the time when all tasks are finished is recorded and averaged over five runs. I only consider end devices for simplicity and all 8 end devices are working by default. I simulate device failures by letting them automatically shutdown after processing half the number of tasks that a worker is going to process, which is 12 in this case. For simplicity, the timeout of transferring data to another device is set to 0.5 (s), which means at most 0.5 (s) delay is generated for each communication with end devices. The curve of averaged duration time with respect to the number of device failure is illustrated in Figure 4, and the raw time data is listed in Table 5.

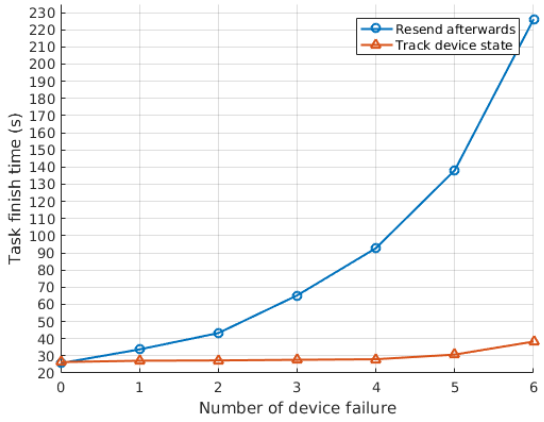

Figure 4: The time to finish the classification task with different methods for fault tolerance, under the circumstances of different number of device failures.

Without any functionality dealing with device failure, the task can not be done if any device fails, which is labeled as 'Inf' in Table 5. With the functionality of reassigning sub-tasks enabled, the system is able to finish the entire task even if some devices fail, but for longer time than that with no device failure. The drawback is that for each communication to the failed device, the time for waiting the network timeout is wasted, causing an unacceptable increasing duration curve. To avoid the timeout overhead in method two, method three maintain a global device list, containing devices currently available. When assigning a sub-task, the next device in the global device list is chosen as the worker. From Table 5, it is observed that this method generates considerably less overhead especially for large number of devices failures, e.g., 5.9x faster with 6 device failures compared with previous method. It is also observed that subtle overhead is generated by applying method two and three, 3.2% and 6.0% longer than method one respectively. However, considering their advantages of enabling fault tolerance for the system, the overhead is acceptable.

Table 5: The time to finish the classification task with different methods for fault tolerance, under the circumstances of different number of device failures.

Methods	Number of device failures						
	0	1	2	3	4	5	6
Without (s)	24.9	Inf	Inf	Inf	Inf	Inf	Inf
Reassign (s)	25.7	33.7	43.2	65.2	92.8	138	226
Monitor (s)	26.4	27.2	27.3	27.7	28.0	30.7	38.3

5 Conclusion

In this paper, a novel framework for the optimization of DL applications on hierarchical distributed systems is proposed. Three types of heterogeneity based on a deeper observation on real-world distributed systems are proposed, (a). distributed computing node heterogeneity, (b). neural network heterogeneity, and (c). system task heterogeneity. A simulated system of proposed framework is being tested on image classification jobs, showing that proposed framework is able to provide rich features on a highly heterogeneous distributed system. Illustrated by elaborately designed experiments, proposed framework offers low response time, optimized resource usage, high performance, better user experience, privacy protection and robustness.

References

[1] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*, 2017, pp. 328–339.

[2] G. Ghimpeteanu, T. Batard, M. Bertalmío, and S. Levine, "A decomposition framework for image denoising algorithms," *IEEE Trans. Image Processing*, vol. 25, no. 1, pp. 388–399, 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition,CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 770–778.

[4] N. Goto, K. Yamamoto, and S. Nakagawa, "English to japanese spoken lecture translation system by using dnn-hmm and phrase-based smt," in *Advanced Informatics: Concepts, Theory and Applications (ICAICTA), 2015 2nd International Conference on.* IEEE, 2015, pp. 1–6.

[5] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech & Language Processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[6] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing," *OJCC*, vol. 2, pp. 16–24, 2015.

[7] 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," *CoRR*, vol. abs/1704.04861, 2017.

[8] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[9] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *J. Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[10] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, 2003, pp. 29–43.

[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, 2012, pp. 15–28.

[12] J. Spillner, J. Müller, and A. Schill, "Creating optimal cloud storage systems," *Future Generation Comp. Syst.*, vol. 29, no. 4, pp. 1062–1072, 2013.

[13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Tech. Rep., 2009.

[14] A. Botta, W. de Donato, V. Persico, and A. Pescapè, "Integration of cloud computing and internet of things: A survey," *Future Generation Comp. Syst.*, vol. 56, pp. 684–700, 2016.

[15] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, 2016.

[16] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[17] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012*, 2012, pp. 13–16.

[18] A. G. Greenberg, J. R. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2009.

[19] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[20] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *The 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'14, Bretton Woods, NH, USA, June 16-19, 2014*, 2014, pp. 68–81.

[21] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *European Conference on Computer Systems, Proceedings of the Sixth European conference on Computer systems, EuroSys 2011, Salzburg, Austria, April 10-13, 2011*, 2011, pp. 301–314.

[22] C. Perera, A. B. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[23] A. Whitmore, A. Agarwal, and L. D. Xu, "The internet of things - A survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[24] A. B. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," *CoRR*, vol. abs/1301.0159, 2013. [Online]. Available: http://arxiv.org/abs/1301.0159

[25] L. W. F. Chaves and C. Decker, "A survey on organic smart labels for the internet-of-things," in *Seventh International Conference on Networked Sensing Systems, INSS 2010, Kassel, Germany, June 15-18, 2010*, 2010, pp. 161–164.

[26] A. M.-H. Kuo, "Opportunities and challenges of cloud computing to improve health care services," *Journal of medical Internet research*, vol. 13, no. 3, 2011.

[27] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Trans. Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[28] M. S. Hossain and G. Muhammad, "Cloud-assisted industrial internet of things (iiot) - enabled framework for health monitoring," *Computer Networks*, vol. 101, pp. 192–202, 2016.

[29] M. M. Rathore, A. Ahmad, A. Paul, and S. Rho, "Urban planning and building smart cities based on the internet of things using big data analytics," *Computer Networks*, vol. 101, pp. 63–80, 2016.

[30] S. Chen, C. Lai, Y. Huang, and Y. Jeng, "Intelligent home-appliance recognition over iot cloud network," in *2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013, Sardinia, Italy, July 1-5, 2013*, 2013, pp. 639–643.

[31] F. Ding, A. Song, D. Zhang, E. Tong, Z. Pan, and X. You, "Interference-aware wireless networks for home monitoring and performance evaluation," *IEEE Transactions on Automation Science and Engineering*, vol. 99, pp. 1–12, 2017.

[32] F. Gao, "Vsaas model on dragon-lab," *International Journal of Multimedia & Ubiquitous Engineering*, vol. 8, no. 4, 2013.

[33] A. Prati, R. Vezzani, M. Fornaciari, and R. Cucchiara, "Intelligent video surveillance as a service," in *Intelligent multimedia surveillance*. Springer, 2013, pp. 1–16.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems, December 3-6, 2012, Lake Tahoe, Nevada, United States.*, 2012, pp. 1106–1114.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, pp. 211–252, 2015.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[37] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, 2016.

[38] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[39] B. McDanel, S. Teerapittayanon, and H. T. Kung, "Embedded binarized neural networks," in *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks, EWSN 2017, Uppsala, Sweden, February 20-22, 2017*, 2017, pp. 168–173.

[40] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, 2012, pp. 1232–1240.

[41] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "Firecaffe: Near-linear acceleration of deep neural network training on compute clusters," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 2592–2600.

[42] P. H. Jin, Q. Yuan, F. N. Iandola, and K. Keutzer, "How to scale distributed deep learning?" in *Machine Learning Systems Workshop on Advances in neural information processing systems*, Dec. 2016.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[44] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.