Received: 24 March 2020 | Revised: 4 August 2020 | Accepted: 24 August 2020

DOI: 10.4218/etrij.2020-0112

SPECIAL ISSUE

ETRI Journal WILEY

Performance analysis of local exit for distributed deep neural networks over cloud and edge computing

Changsik Lee | Seungwoo Hong | Sungback Hong | Taeyeon Kim

Hyper-connected Communication, Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea

Correspondence
Changsik Lee, Hyper-connected, Communication Research Laboratory, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea.
Email: cslee2624@etri.re.kr

Funding information
ICT R&D Program of MSIT/IITP, Rep. of Korea, Grant No. 2018-0-01502.

In edge computing, most procedures, including data collection, data processing, and service provision, are handled at edge nodes and not in the central cloud. This decreases the processing burden on the central cloud, enabling fast responses to end-device service requests in addition to reducing bandwidth consumption. However, edge nodes have restricted computing, storage, and energy resources to support computation-intensive tasks such as processing deep neural network (DNN) inference. In this study, we analyze the effect of models with single and multiple local exits on DNN inference in an edge-computing environment. Our test results show that a single-exit model performs better with respect to the number of local exited samples, inference accuracy, and inference latency than a multi-exit model at all exit points. These results signify that higher accuracy can be achieved with less computation when a single-exit model is adopted. In edge computing infrastructure, it is therefore more efficient to adopt a DNN model with only one or a few exit points to provide a fast and reliable inference service.

KEYWORDS

convolutional neural networks, deep neural networks, edge computing, inference performance, multi-exit, single-exit

1 | INTRODUCTION

Recently, deep neural networks (DNNs) have received considerable attention owing to their high accuracy and reliable results. They have been utilized in numerous applications such as computer vision [1], speech recognition [2], natural language processing [3], and network traffic classification [4]. Traditionally, application services based on DNN models have been executed in a central cloud server as they require enormous computing and memory resources. In a central cloud architecture, raw data generated at end devices are delivered to a central cloud for preprocessing and are processed through a DNN model. The inference result is then provided to the users as a service.

In a central cloud architecture, the cloud simultaneously handles big data, such as images, videos, and audio recordings, generated from various end devices, such as cars, sensors, cameras, and Internet-of-Things (IoT) devices. This leads to prolonged response times and exhaustion of the network bandwidth between the cloud and end devices. Moreover, the amount of data worldwide is expected to reach 163 ZB, which is 10 times more than it is now, owing to numerous IoT artificial intelligence (AI) services and an explosive increase in the number of smart devices [5].

To address this impending problem, edge computing is regarded as a promising infrastructure that can provide efficient services to end devices. In addition, edge computing alleviates the processing burden of a central cloud. In an edge computing infrastructure, most of the procedures, including data collection, information processing, and service provision, are handled at edge nodes (eg, a server attached to an access point or a network gateway), decreasing the processing burden on the central cloud. Because the edge nodes are distributed near the end devices, response times to end-device service requests are short and bandwidth consumption is reduced. Recently, edge computing has been utilized as an infrastructure for ultra-low-latency services such as self-driving cars, the Tactile Internet, virtual reality, and augmented reality.

However, in contrast to cloud computing, the edge nodes in edge computing have limited computing, storage, and energy resources. Therefore, edge computing has practical challenges in providing AI services that entail computation-intensive tasks such as processing a DNN inference. To overcome these challenges, many studies have been conducted that focus on improving the efficiency of DNN inference [6,7]. For example, for fast and low-power DNN inference at edge nodes, network compression schemes and DNN architecture optimization have been proposed [8–12]. These network compression schemes aim to reduce the total number of DNN model parameters and thus minimize the amount of computation required to perform inference.

Recently, several researchers have proposed methods that partition DNN computation between the central cloud and mobile devices at the granularity of neural network (NN) layers for collaborative intelligence between the central cloud and mobile edge. For example, [13] proposed a lightweight scheduler that automatically identifies the ideal partition points in DNNs and orchestrates the distribution of computation between the mobile device and the cloud server. Reference [14] proposed distributing DNNs over distributed computing hierarchies consisting of the cloud, edge, and end devices. The authors adopted a local exit mechanism supported by the open source framework BranchyNet [15]. Using this local exit mechanism, they classify the samples and exit them locally at the edge or end device when the inference result is confident. Moreover, they offload the rest of the samples to the cloud when additional processing is required. Reference [16] presented a collaborative DNN co-inference framework using end devices and edge nodes that jointly optimizes DNN partitioning and right-sizing through local exiting in an on-demand manner. The researchers also demonstrated the effectiveness of their proposed framework via an implementation and evaluations on a Raspberry Pi.

The existing approaches aim to reduce DNN computation while minimizing any loss of accuracy. However, they do not consider the effects of a local exit on inference performance in terms of accuracy and latency. In particular, the appropriate number of exit points to achieve a fast and reliable inference result is not considered. Understandably, a DNN model with multiple exit points has the distinction that it can dynamically select an optimal exit point from among various exit points without the model requiring any modifications. However, it imposes additional computational complexity on the edge to find the optimal exit point and the complexity increases in proportion to the number of available exit points.

To mitigate these concerns, we analyzed the performance of local exits for a distributed DNN over the cloud and edge. Specifically, we generated eight convolutional neural network (CNN) models with single exit points, each with different exit point locations. (We refer to these models as single exit models (SEMs).) We also generated a CNN model with eight exit points, referred to as the multi-exit model (MEM), with the locations of the exit points corresponding to those of the SEMs.

For the performance evaluation, we established a test environment using Docker containers consisting of a test node, an edge node, and a cloud node. Our test results show that the SEMs performed better than the MEM at all exit points in all aspects, including the number of local exit samples, inference accuracy, and inference latency. These results indicate higher accuracy can be achieved with less computation if we adopt a DNN model with a single exit point rather than a model with too many exit points. In other words, to provide fast and reliable inference service using a DNN model in an edge computing infrastructure, it is most efficient to adopt a model with one or only a few exit points. Moreover, from the performance results, we confirmed the need for a specific mechanism to handle difficult samples in order to efficiently exploit the resources for other, easier samples and provide low-latency service.

The remainder of this paper is organized as follows: Section 2 provides background on aspects of the present topic including edge computing, CNNs, and local exits for distributed DNNs. Section 3 introduces the methodology used for local exit evaluation. Section 4 describes the performance test conducted and analyzes the results obtained. Finally, concluding remarks are presented in Section 5.

2 | BACKGROUND

2.1 | Edge computing

In an edge computing infrastructure, most of the procedures are handled at edge nodes—including data collection, data processing, and service provision—decreasing the processing burden on the central cloud. Because edge nodes are distributed near end devices, fast responses are achieved to end-device service requests with reduced bandwidth consumption.

As edge computing has recently been receiving more attention, various standard group and network vendors have begun to adopt it. Mobile edge computing (MEC) standardization is in progress by the European Telecommunications Standards Institute (ETSI) to add edge computing functionality to 5G networks [17–19]. Additionally, the International Telecommunication Union-Telecommunication

(ITU-T) standardization sector has started to develop intelligent edge computing with machine learning to support AI [20]. Furthermore, various edge computing initiatives, such as the OpenFog Consortium driven by Cisco [21,22], Open Edge Computing Initiative [23–25], Edge Computing Consortium [26], and Industrial Internet Consortium [27], have recently begun in the industry.

2.2 | Convolutional neural networks

A standard CNN consists of several NN layers such as convolution layers, normalization layers, pooling layers, activation layers, and fully connected layers. The convolution layers extract simple feature maps from input data by executing convolution operations with convolutional filters. Next, extracted feature maps are processed through activation layers (such as rectified linear units (ReLUs)), and then, the size of the feature maps is decreased through normalization and pooling operations. By repeating these procedures, the CNN model captures a high-level representation of the input data and that is then forwarded to fully connected layers to return the inference result. Various CNN models have recently been proposed to improve performance in computer vision tasks such as image classification (eg, LeNet [28], AlexNet [29], VGGNet [30], GoogLeNet [31], and ResNet [32]) and object detection (eg, RCNN [33], Fast RCNN [34], Faster RCNN [35], SPP Net [36], and YOLO [37]).

2.3 | Local exit for distributed DNN

Reference [15] proposed a solution to classify input samples at earlier points in a NN, called local exit points, using an entropy-based confidence criterion. In the proposed solution, if a sample is deemed to be confident at a local exit point, based on the entropy of the computed probability vector for target classes, then it is locally classified. In this case, the higher NN layers perform no further computation. In previous work [14], exit points could be placed at physical boundaries (such as between the last NN layer in an end device and the first NN layer in the next higher node of the distributed computing hierarchy, namely the edge or the cloud). Using the local exit framework, input samples that can be confidently classified will exit locally, thereby achieving a fast response and reducing network communication to the next physical boundary.

3 | METHODOLOGY FOR LOCAL EXIT EVALUATION

In this section, we describe the design of the local exit framework and then explain the model training and model inference procedures. Figure 1 shows the standard CNN model architecture without local exit, which is traditionally deployed at the central cloud. After the input layer, samples are processed through the convolutional blocks for feature extraction. Each convolutional block includes two repetitions of the following: convolutional layer + normalization layer + activation layer. In the activation layer, we use the ReLU function. Finally, the extracted features are processed through the fully connected block consisting of an average pooling layer and a fully connected layer to compute the probability vector for target classes. The class with the highest probability is decided as the final inference result, herein referred to as cloud exit because the result is the outcome of the last NN layer.

3.1 | Design of local exit evaluation framework

Figure 2 shows the architecture of SEMs where the local exit framework is applied to the standard CNN model. We generated eight models, SEM-1 to SEM-8, with each model having its own exit point at a different location. To inference at the exit point, we locally added the fully connected block after the previous convolutional block. Aside from the location of

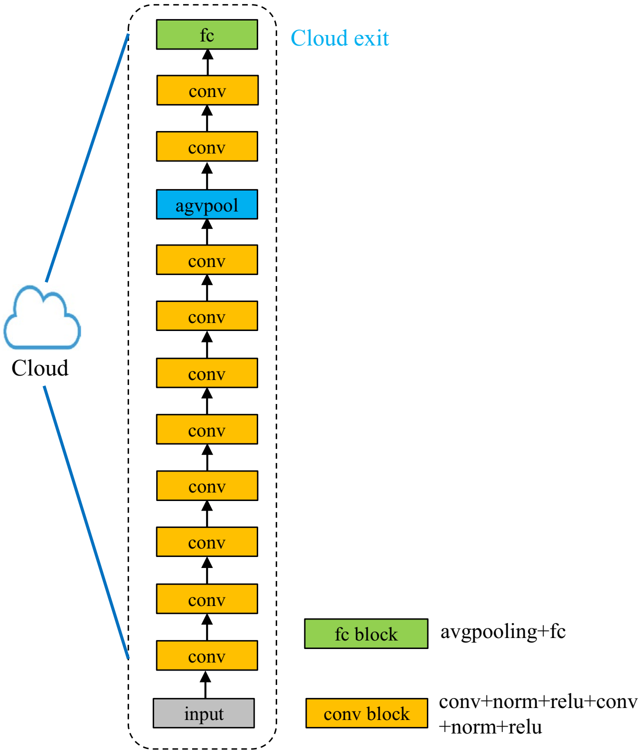

FIGURE 1 Standard CNN model: After the input layer, samples are processed through the convolutional blocks for feature extraction. The size of the feature maps is decreased through normalization and average pooling operations. Finally, the extracted features are processed through the fully connected layer to compute the probability vector for target classes

the local exit point, other configurations such as model size and the location of the cloud exit were the same.

Figure 3 shows the MEM with eight local exit points and one cloud exit point. In the MEM, the location of each local exit point corresponds to that of one of the SEMs. For example, the location of exit point 1 in the MEM is the same as the location of the exit point in SEM-1. Notably, the MEM has the distinction that it can dynamically select the optimal exit point from among various exit points without requiring modifications. The optimal exit point depends on the environment, such as the current status of computing resources, service requirements, or network bandwidth between the cloud and edge.

3.2 | Model training of distributed DNN

We trained each model on a single powerful server. In the training phase, all samples were classified at each exit point, but were also forwarded to the next layer without locally exiting. Then, the losses from all the local exits and the cloud exit were combined during backpropagation so that the entire network could be jointly trained. We used the stochastic gradient descent algorithm as the optimizer and the cross-entropy loss function for exit point losses. In this study, we allocated equal weights to the losses from each exit point. Other training parameters are shown in Table 1. We used the CIFAR10 dataset for model training and performance evaluation.

3.3 | Model inference of distributed DNN

In this section, we describe the model inference procedure. First, input images are processed through several convolutional blocks until the exit point for feature extraction. Subsequently, the probability vector for target classes is computed via the local fully connected block. Based on the probability vector at the exit point, normalized entropy is computed as a measure of confidence in the prediction. We followed the description of normalized entropy from [14].

$$\varepsilon(\mathbf{p}) = -\sum_{i=1}^{|L|} \frac{p_i \log p_i}{\log |L|}, \tag{1}$$

where L is the set of all possible labels and $\mathbf{p}$ is a probability vector. Entropy ε has values between zero and one. For example, ε close to zero means that it is confident about the inference of the sample, whereas ε close to one means it is not confident.

The computed entropy is compared against the exit threshold (T) to determine whether the sample should be exited at that exit point or not. If the entropy is smaller than the exit threshold (ie, $\varepsilon < T$), the inference result is reliable, and thus, the sample is classified (local exit). Conversely, if the entropy is larger than the exit threshold (ie, $\varepsilon > T$), the result of the intermediate computation output from the previous convolutional block is sent to the next convolutional block for further processing. Then, the model performs the final inference at the last NN layer (cloud exit). For exit procedures such as local exit or cloud exit, the class with the highest probability is determined to be the prediction result. Note that at the inference phase in the MEM, only one of eight exit points was set up to execute local exit in order to compare its performance under the same condition as that of the SEMs.

4 | PERFORMANCE EVALUATION

4.1 | Test environment setup

Figure 4 shows the test environment topology based on a Docker container consisting of three nodes: a cloud node, an edge node, and a test node. In performance testing, the test node sends image samples to the edge node with batch size 32. The edge node then proceeds with computation to the exit point to determine whether the sample can exit locally. If at a local exit point a sample is deemed confident based on the entropy of the computed probability vector for target classes, then it is locally classified, and the edge node sends the input image's predicted class to the test node. Otherwise, the edge node forwards the intermediate computation to the cloud node for further processing. Then, the cloud node executes the remaining layers and sends the predicted class of the input image to the test node. Finally, the test node obtains the classification results from the edge node and cloud node to calculate inference accuracy and latency.

To train our models, we utilized a separate training server with a Nvidia GeForce 1080Ti and graphic driver v390.116. Further, we carried out all performance tests by varying the exit threshold (ie, 0.1, 0.3, 0.5, and 0.8). Note that the opportunity for local exit depends on the value of the exit threshold. Although each model's performance changed with the exit threshold, we confirmed a similar performance tendency associated with the exit threshold among the models. In this study, we demonstrate the performance evaluation with a fixed exit threshold ($T = 0.3$). The best value for the exit threshold depends on the NN architecture or dataset; however, determination of this best value is beyond the scope of this study.

The different accuracy and latency measures associated with the performance evaluation for the local and cloud exits are defined as follows:

- Edge accuracy is the accuracy of the samples that exit at local exit points.
- Cloud accuracy is the accuracy of the samples that exit at the cloud node.

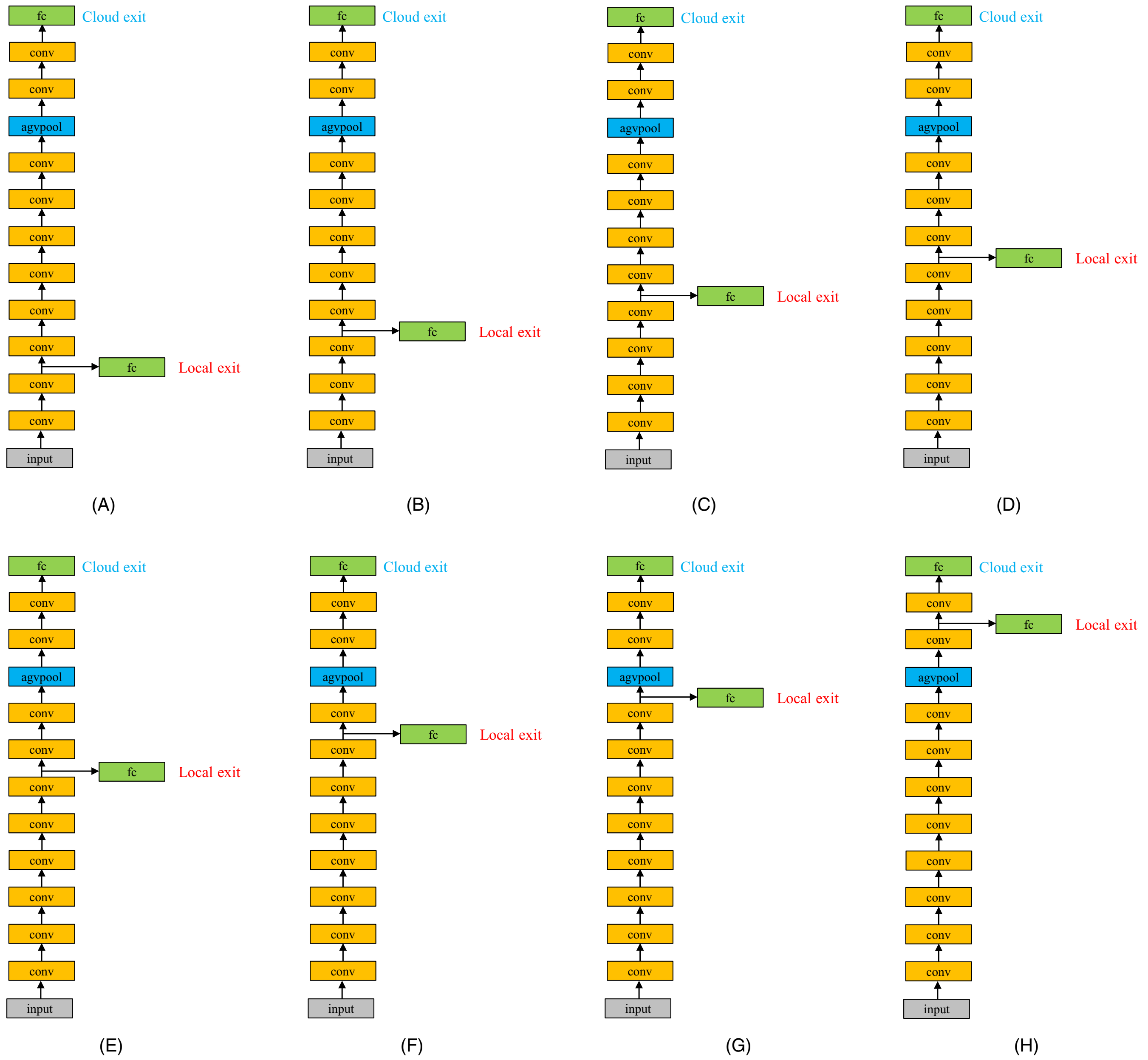

FIGURE 2 Eight CNN models with single exit points. Each model has its exit point at a different location from that of the others: (A) SEM-1, (B) SEM-2, (C) SEM-3, (D) SEM-4, (E) SEM-5, (F) SEM-6, (G) SEM-7, and (H) SEM-8

- Overall accuracy is the average accuracy of all the test samples. It is calculated as (sum of the corrected samples)/(total number of samples).
- Edge latency is the average inference latency of the samples that exit at local exit points. It includes the time for the procedure 1) → 2) → 3-1) in Figure 4.
- Cloud latency is the average inference latency of the samples that exit at the cloud node; in other words, those that do not exit locally. It includes the time for the procedure 1) → 2) → 3-2) → 4)→ 5) in Figure 4.
- Overall latency is the average inference latency of all the test samples. It is calculated as the time duration from sending the first image until receiving the predicted class of the last image.

4.2 | Performance of SEMs

In this section, we analyze the inference performance of each SEM, such as the number of samples exited locally, the inference accuracy, and the inference latency. As shown in Figure 5A, models with exit points in the rear exited locally more frequently. In particular, for SEM-3, the number of samples that exited locally (5383 samples) exceeded the number of cloud-exited samples (4617 samples). This signifies that models with exit points in the rear exhibit more confident classification results. This is reasonable because such models perform further computation through more NN layers until their local exit point.

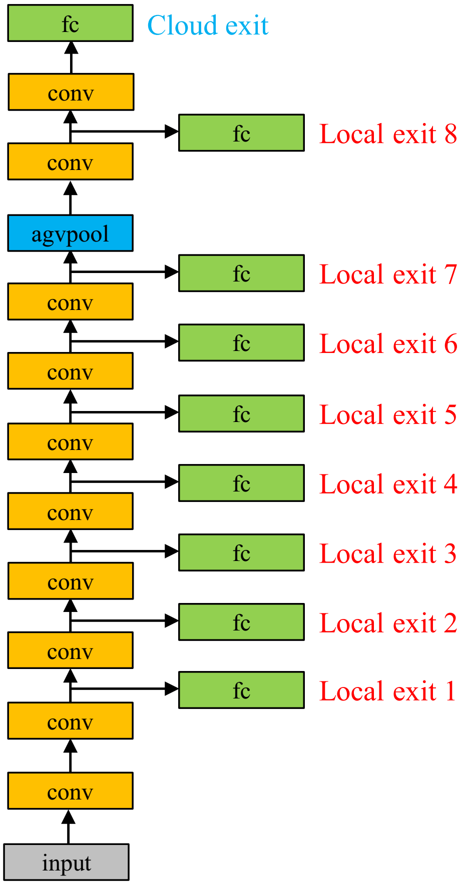

FIGURE 3 MEM with eight exit points. The locations of the exit points correspond to those of the SEMs

TABLE 1 Training parameters

Parameters	Value
Learning rate	0.1
Momentum	0.9
Weight decay	10^{-4}
Batch size	32
Training epoch	2000
Number of training samples	50 000
Number of test samples	10 000

Figure 5B shows the accuracy performance of each model. For the abovementioned reason, models with exit points in the rear show higher edge accuracy (eg, 87.20% in SEM-1 as compared with 95.24% in SEM-8). However, they tend to perform worse in terms of cloud accuracy because some test samples are difficult to classify accurately even with extensive computation. Thus, these samples are not exited locally and are forwarded to the cloud node. For example, in SEM-8, a few samples (861 samples) are not exited locally even though they are processed through most of the NN layers. Thus, these samples are unlikely to be correctly classified through just one more convolutional block, leading to a low cloud accuracy (58.07%). In contrast, in models with local exit points in the front, such as SEM-1 and SEM-2, samples that are not exited locally have a high probability of being correctly classified through sufficient computation in the cloud node. Notably, these samples lead to a higher cloud accuracy (88.51% and 86.13%, respectively) than that of SEM-8. However, this entails additional communication latency and computational latency in the cloud node as well as additional energy consumption of the edge node to enable further processing in the cloud node. This signifies that in the case of difficult samples, resources such as computing, networking, storage, and energy are wasted when processing is further performed in the cloud node. Thus, we need a specific mechanism to handle them (eg, to drop a sample if the entropy is over the drop threshold) in order to efficiently use the resources for other easier samples and provide low-latency service.

Overall accuracy is the measure of how well the model is jointly optimized for local exit and cloud exit losses during model training. As shown in Figure 5B, models with exit points in the rear tended to show higher overall accuracy. For example, SEM-3 showed the worst overall accuracy (86.04%) whereas SEM-8 showed the best overall accuracy (92.04%).

Figure 5C shows the inference latency performance of each model. SEM-1 showed the shortest edge latency because it has the smallest number of computational layers until its exit point. Nevertheless, as shown in Figure 5A, it also has the smallest

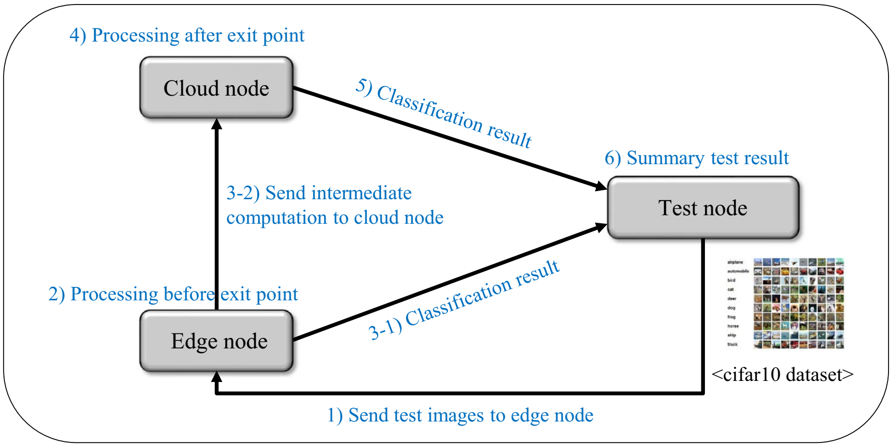
4) Processing after exit point
Cloud node
5) Classification result
6) Summary test result
Test node
3-2) Send intermediate computation to cloud node
2) Processing before exit point
3-1) Classification result
Edge node
<cifar10 dataset>
1) Send test images to edge node

FIGURE 4 Test environment topology based on a Docker container. It consists of three nodes: a cloud node, an edge node, and a test node

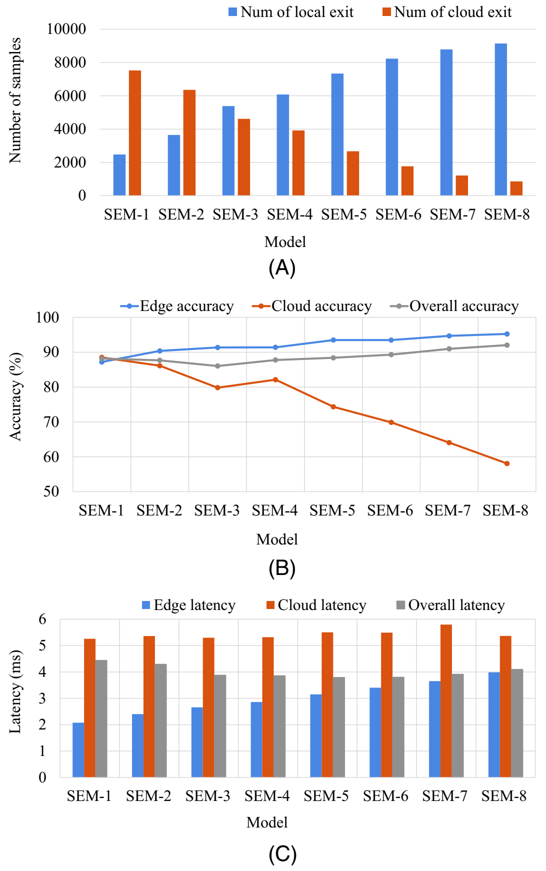

FIGURE 5 Inference performance of SEMs: (A) Number of samples exited locally, (B) inference accuracy, and (C) inference latency

number of samples exited locally, as most of the samples are exited on the cloud node. Consequently, SEM-1 exhibits the longest overall inference time (gray bar). In respect of cloud latency, all models showed similar performance as they have the same NN layer depth until the cloud exit point. Although SEM-5 showed the shortest overall latency (3.795 ms) among SEMs, this does not mean that it guarantees the best latency in a different NN model or dataset. Furthermore, overall latency performance would be affected by a dynamically changing network bandwidth between the edge node and cloud node, leading to a change in the optimal exit point.

4.3 | Performance of MEM

In this section, we evaluate the inference performance of the MEM with eight local exit points. As mentioned in Section 3.2, during the model training, all samples are classified at each exit point but are also forwarded to the next layer without locally exiting. Then, the losses from the eight local exits and the cloud exit are combined during backpropagation so that the entire network is jointly trained. For model inference, one of the eight exit points is set up to perform the local exit.

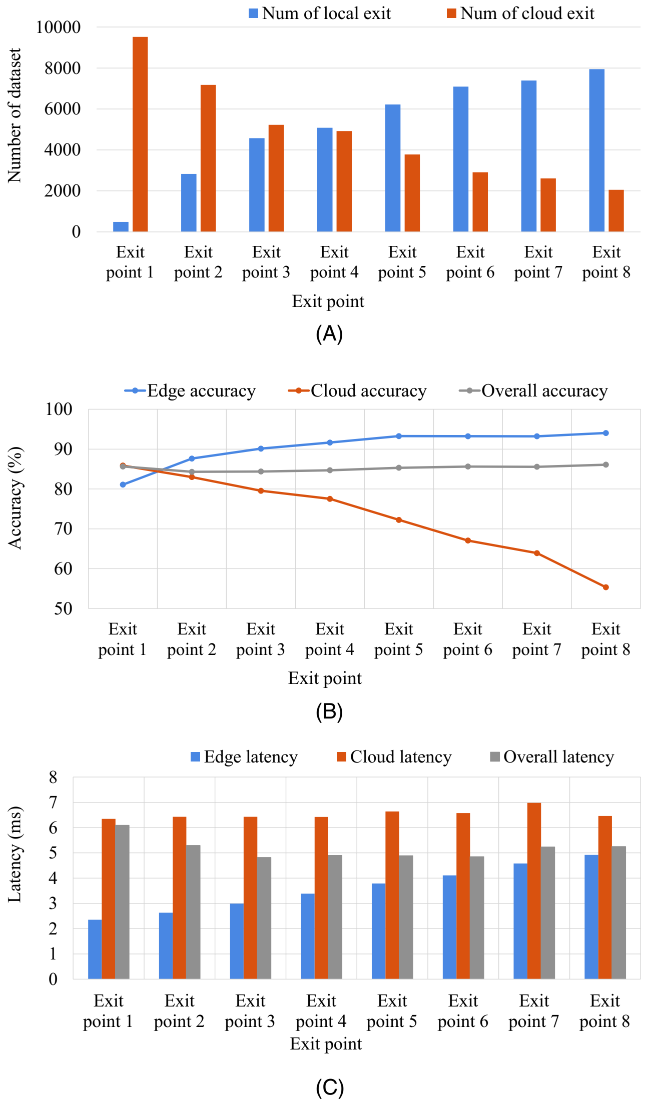

FIGURE 6 Inference performance of the MEM: (A) Number of samples exited locally, (B) inference accuracy, and (C) inference latency

Figure 6 shows the inference performance of the MEM when varying the location of the exit point. Generally, the MEM shows similar performance tendencies as those of the SEMs. However, it has a smaller number of locally exited samples and a lower inference accuracy compared with the SEM at the corresponding exit point. In overall accuracy, it notably showed the worst performance (84.31%) at exit point 2 and the best performance (86.10%) at exit point 8. This signifies that the existence of needlessly many local exit points impairs the optimality of the entire NN during model training.

Figure 6C shows the inference latency of the MEM when the location of the activated exit point is varied. The MEM shows the shortest overall latency (4.82 ms) at exit point 3, whereas it shows an overall latency of 4.886 ms at exit point 5. Note that exit point 5 is the same point at which the SEM

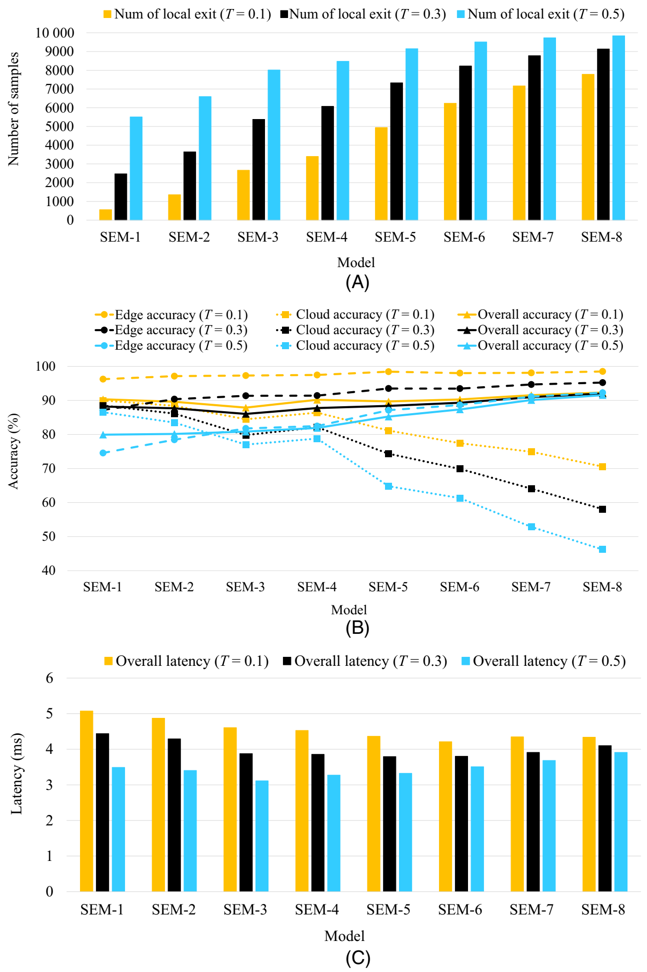

FIGURE 7 Inference performance of the SEMs for different exit thresholds ($T = 0.1$, $T = 0.3$, and $T = 0.5$): (A) Number of samples exited locally, (B) inference accuracy, and (C) overall latency

exhibited the shortest overall latency of 3.795 ms (in SEM-5). As expected, the reason why the MEM showed longer overall latency than the SEM at the corresponding exit point is that the MEM has a smaller number of samples exited locally than the SEM, as shown in Figure 6A.

From the performance results, we verified that the excessive number of local exit points in the DNN model leads to performance degradation such as low inference accuracy and slow inference time. The overall accuracy becomes particularly worse (from 92.04% to 86.10%) at exit point 8. This result is owing to the fact that for training the MEM, the losses from the eight local exits and the cloud exit are combined during backpropagation so that the entire network is jointly trained. That is, model weights are updated for the optimization of all exit points, not for a specific exit point. In contrast, model weights are updated for the optimization of only one exit point and the cloud exit during the training of the SEM. Therefore, SEM exhibits a higher inference accuracy at a specific exit point than that exhibited by MEM after model training.

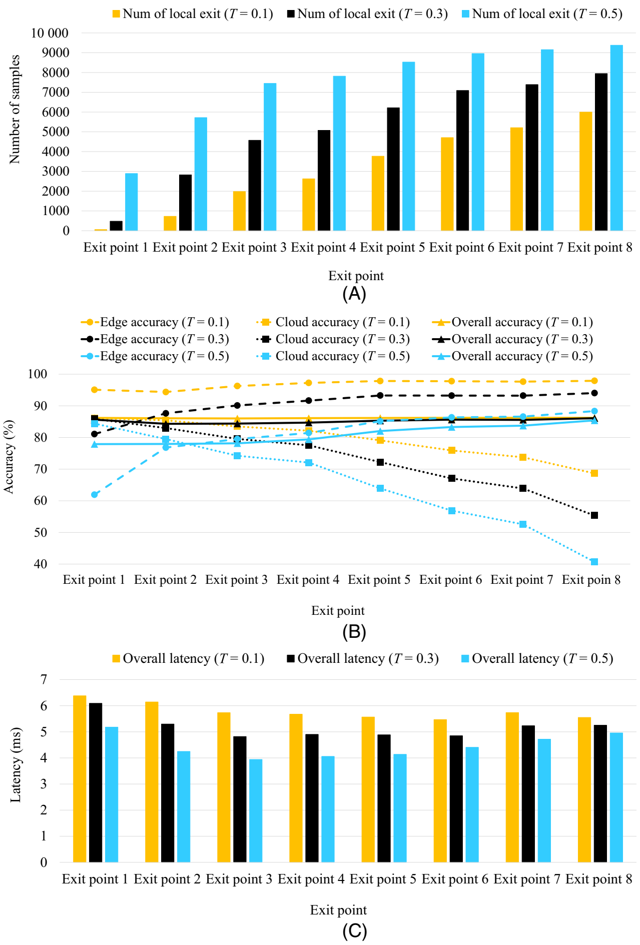

FIGURE 8 Inference performance of MEM for different exit thresholds ($T = 0.1$, $T = 0.3$, and $T = 0.5$): (A) Number of samples exited locally, (B) inference accuracy, and (C) overall latency

4.4 | Performance comparison for different exit thresholds

In this section, we present the analysis of the inference performance of each model for different exit thresholds such as $T = 0.1$, 0.3, and 0.5. Figure 7 shows the performance results of the SEMs. As shown in Figure 7A, samples have an increased possibility of being locally exited under a higher exit threshold ($T = 0.5$), whereas fewer samples are locally exited under a lower exit threshold ($T = 0.1$). Figure 7B shows the inference accuracy with respect to different exit thresholds. All the accuracy metrics, including edge accuracy, cloud accuracy, and overall accuracy, increase as the exit threshold decreases. Notably, the edge accuracy and overall accuracy show the largest performance gap in SEM-1. These results are reasonable because a low exit threshold requires more confidence regarding the prediction to exit samples locally. That is, only samples with extremely high confidence are locally exited under a lower exit threshold ($T = 0.1$), and most of the samples are sent to the cloud for further processing. In contrast, some samples with uncertainty are locally exited under a higher exit

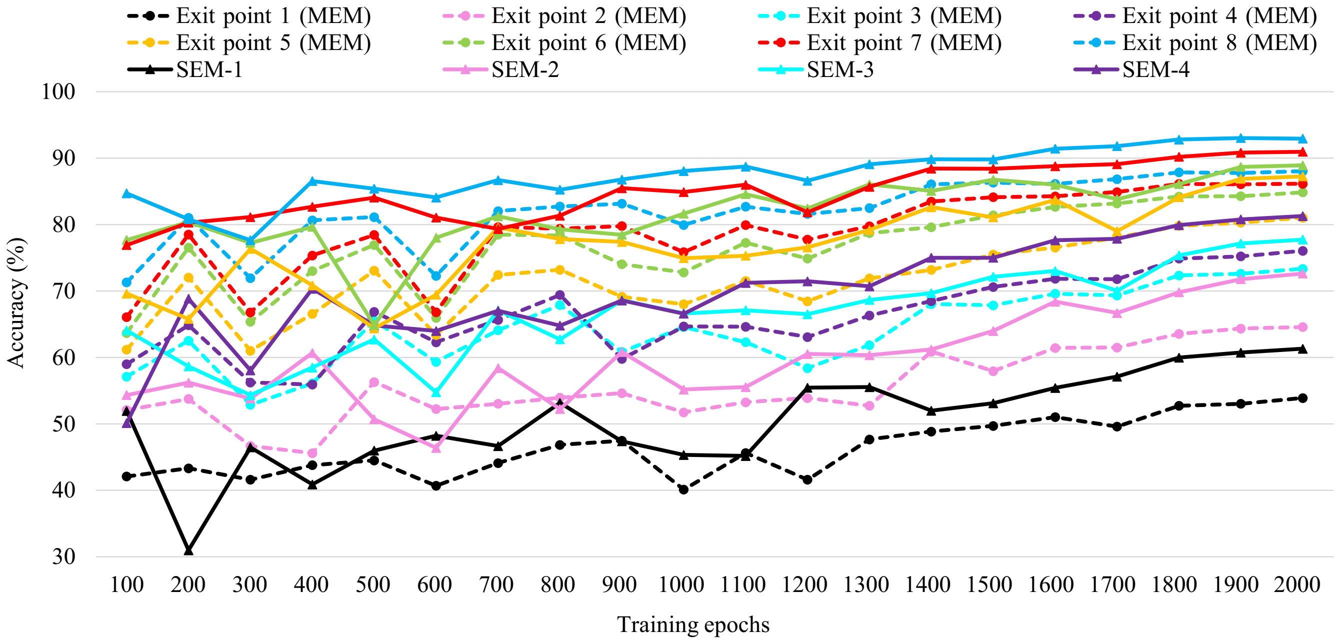

FIGURE 9 Accuracy comparison at each exit point of the SEMs and MEM over training epochs

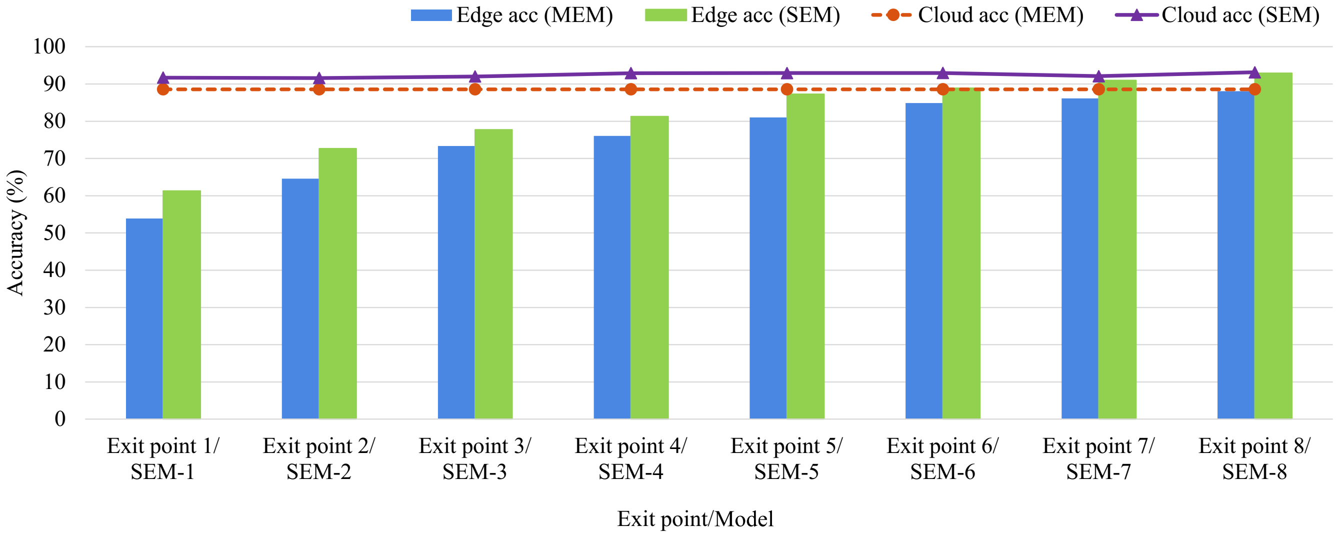

FIGURE 10 Edge accuracy and cloud accuracy of the SEMs and MEM at the end of model training

threshold ($T = 0.5$), leading to low accuracy. Additionally, as shown in Figure 7C, the overall inference latency increases as the exit threshold decreases. Furthermore, we verified that the MEM also shows similar performance results as the exit threshold is varied, as shown in Figure 8.

4.5 | Comparison of training accuracy

In this section, we compare the training accuracy of the SEMs and MEM. Note that in the training phase, all samples are classified at each exit point, but are also forwarded to the next layer without locally exiting. Figure 9 shows the accuracy of the SEMs and MEM at each exit point as the number of training epochs increase. In general, the accuracy of the SEM (solid line) is higher than that of the MEM (dotted line) in a corresponding exit point location. At the end of training, the SEM with an earlier exit point even outperformed the MEM with a later exit point (eg, SEM-3 outperforms the MEM with exit point 4).

Figure 10 shows the edge accuracy and cloud accuracy at the end of the training phase. The lines signify cloud accuracy, whereas the bars signify edge accuracy. Note that the cloud accuracy of the MEM is fixed at 88.58%. In both accuracy metrics, the SEM outperformed the MEM at the corresponding exit points. In particular, the edge accuracy showed the largest performance gap at exit point 2 (SEM-2); for example, 72.65% in SEM-2 and 64.59% in the MEM. Additionally, as mentioned above, the edge accuracy of SEM-3 (77.76%) was higher than the edge accuracy of the MEM with exit point 4 (76.06%). Notably, this performance gap increased as the exit points moved toward the rear. Furthermore, we verified that the edge accuracy of the SEM after exit point 6 surpassed even the cloud accuracy of the MEM. For example, the edge accuracy of SEM-6 was 88.92%, whereas the cloud accuracy of the MEM was 88.58%. These results indicate higher accuracy can be achieved with less computation if we adopt a DNN model with single exit points rather than too many exit points. Furthermore, after SEM-7, edge accuracy was even closer to the cloud accuracy in the same model. Therefore, further processing is not required in the cloud node. For example, the edge accuracy and cloud accuracy of SEM-7 were 90.98% and 92.10%, respectively. As mentioned in Section 4.2, this result supports the need of a specific mechanism to handle difficult samples.

5 | CONCLUSIONS

In this study, we analyzed the effect of local exit on the inference performance of DNN models in an edge computing environment. We designed several DNN models by varying the location and number of exit points for performance evaluation. The performance test results demonstrate that a higher accuracy is achieved with less computation if we adopt a DNN model with a single exit point rather than one with an excessive number of exit points. Consequently, with regard to providing an inference service with low latency and high accuracy based on a DNN model in an edge computing infrastructure, adopting a model with one or only a few exit points is more efficient. Moreover, the performance results indicate the need for a specific mechanism to handle difficult samples to efficiently use the resources to infer other, easier samples and provide a low-latency service. In this study, we allocated equal weights for the loss from each exit point during model training. In the future, we will conduct performance evaluation by assigning different weights for the losses. Furthermore, we will evaluate performance under a practical environment wherein the edge nodes have restricted computing resources.

ACKNOWLEDGEMENTS

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [2018-0-01502, A Development for Intellectualized Edge Networking based on AI].

ORCID

Changsik Lee https://orcid.org/0000-0002-3825-7317

REFERENCES

1. C. Szegedy et al., *Going deeper with convolutions*, in Proc. IEEE Conf. Comput. Vision Pattern. Recogn. (Boston, MA, USA), June 2015, pp. 1–9.
2. A. van den Oord et al., *WaveNet: A generative model for raw audio*, in Proc. ISCA Speech Synthesis Workshop (Sunnyvale, CA USA), Sept. 2016.
3. D. Wang and E. Nyberg, *A long short-term memory model for answer sentence selection in question answering*, in Proc. Annu. Meeting Ass. Comput. Linguistics Int. Joint Conf. Natural Language Process (Beijing, China), July 2015, pp. 707–712.
4. M. Kim, *Supervised learning-based DDoS attacks detection: tuning hyperparameters*, ETRI J. **41** (2019), 560–573.
5. D. Reinsel, J. Gantz, and J. Rydning, *The digitization of the world from edge to core*, White Paper US44413318, IDC (Framingham, MA, USA), Nov. 2018, pp. 1–28.
6. M. Murshed et al., *Machine learning at the network edge: A survey*, arXiv preprint, 20192019, arXiv:1908.00080.
7. Z. Zhou et al., *Edge intelligence: paving the last mile of artificial intelligence with edge computing*, Proc. IEEE **107** (2019), 1738–1762.
8. S. Han, H. Mao, and W. Dally, *Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding*, in Proc. Int. Conf. Learn. Representations (San Juan, Puerto Rico), May 2016.
9. Y. Kim et al., *Compression of deep convolutional neural networks for fast and low power mobile applications*, in Proc. Int. Conf. Learn. Representations (San Juan, Puerto Rico), May 2016.
10. J. Wu et al., *Quantized convolutional neural networks for mobile devices*, in Proc. IEEE Conf. Comput. Vis. Pattern Recogn. (Las Vegas, NV, USA), 2016, pp. 4820–4828.
11. N. Lane et al., *Dxtk: Enabling resource-efficient deep learning on mobile and embedded devices with the deepx toolkit*, in Proc. MobiCASE (Cambridge, UK), Dec. 2016, pp. 98–107.
12. F. N. Iandola et al., *Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size*, arXiv preprint, 2016, arXiv:1602.07360.
13. Y. Kang et al., *Neurosurgeon: Collaborative intelligence between the cloud and mobile edge*, in Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst. (Xian, China), Apr. 2017, pp. 615–629.
14. S. Teerapittayanon, B. McDanel, and H. T. Kung, *Distributed deep neural networks over the cloud, the edge and end devices*, in Proc. IEEE Int. Conf. Distrib. Comput. Syst. (Atlanta, GA, USA), June 2017, pp. 328–339.
15. S. Teerapittayanon, B. McDanel, and H. Kung, *Branchynet: Fast inference via early exiting from deep neural networks*, in Proc. Int. Conf. Pattern Recogn. (Cancun, Mexico), Dec. 2016, pp. 2464–2469.
16. E. Li, Z. Zhou, and X. Chen, *Edge intelligence: On-demand deep learning model co-inference with device-edge synergy*, in Proc. Workshop Mobile Edge Commun. (Budapest, Hungary), Aug. 2018, pp. 31–36.
17. ETSI, *Executive Briefing–Mobile Edge Computing (MEC) Initiative*, Sept. 2014.
18. ETSI Gs MEC-IEG 004, *Mobile Edge Computing (MEC) Service Scenarios V1.1.1*, 2015.
19. ETSI Gs MEC 003, *Mobile Edge Computing (MEC) Framework and Reference Architecture V1.1.1*, 2016.
20. ITU-T Rec. Y.3172, *Architectural framework for machine learning in future networks including IMT-2020*, 2019.
21. OpenFog Consortium Architecture Working Group, *OpenFog architecture overview*, Open fog consortium (Tokyo, Japan), White Paper OPFWP001.0216, Feb. 2016.
22. OpenFog Consortium Architecture Working Group, *Openfog reference architecture for fog computing, Open fog consortium* (Tokyo, Japan), Feb. 2017.

23. M. Satyanarayanan, *The emergence of edge computing*, Comput. **50** (2017), 30–39.
24. Z. Chen et al., *An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance*, in Proc. ACM/IEEE Symp. Edge Comput. (San Jose, CA, USA), Oct. 2017, pp. 1–14.
25. J. Wang et al., *Towards scalable edge-native applications*, in Proc. ACM/IEEE Symp. Edge Comput. (Rlington, VA, USA), 2019, pp. 152–165.
26. Edge Computing Consortium, *White paper of edge computing consortium*, ECC (Beijing, China), White Paper, Nov. 2016.
27. S.-W. Lin et al., *Industrial internet reference architecture*, Ind. Internet Consortium (IIC) (Needham, MA, USA) Tech. Rep., June. 2015.
28. Y. LeCun et al., *Gradient-based learning applied to document recognition*, Proc. IEEE **86** (1998), 2278–2324.
29. A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, in Proc. Conf. Neural Inf. Process Syst. (Stateline, NV, USA), 2012, pp. 1097–1105.
30. K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint, 2014, arXiv:1409.1556.
31. C. Szegedy et al., *Going deeper with convolutions*, in Proc. IEEE Conf. Comput. Vision Pattern Recogn. (Boston, MA, USA), June 2015, pp. 1–9.
32. K. He et al., *Deep residual learning for image recognition*, arXiv preprint, 2015, arXiv:1512.03385, 2015.
33. R. Girshick et al., *Rich feature hierarchies for accurate object detection and semantic segmentation*, in Proc. IEEE Conf. Comput. Vision Pattern Recogn. (Columbus, OH, USA), June 2014, pp. 580–587.
34. R. Girshick, *Fast R-CNN*, in Proc. IEEE Int. Conf. Comput. Vision (Santiago, Chile), Dec. 2015, pp. 1440–1448.
35. S. Ren, *Faster R-CNN: Towards real time object detection with region proposal networks*, in Proc. Int. Conf. Neural Inf. Process. Syst. (Montreal, Canada), 2015, pp. 91–99.
36. K. He et al., *Spatial pyramid pooling in deep convolutional networks for visual recognition*, in Proc. Eur. Conf. Comput. Vis. (Zurich, Switzerland), Sept. (2014), 346–361.
37. J. Redmon, et al., *You only look once: Unified, real time object detection*, in Proc. IEEE Conf. Comput. Vision Pattern Recogn. (Las Vegas, NV, USA), June 2016, pp. 779–788.

AUTHOR BIOGRAPHIES

Changsik Lee received his BS degree in electrical engineering from Korea University, Seoul, Rep. of Korea, in 2012, and his MS degree in electrical engineering from the Korean Advanced Institute of Science and Technology, Daejeon, Republic of Korea, in 2014. Since 2014, he has been with the Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea, where he is currently a researcher. His interests include virtual router redundancy, OpenFlow, edge computing, and machine learning algorithms.

Seungwoo Hong received his MS degree in computer science from Pusan National University, Pusan, Rep. of Korea, in 2001, and his PhD degree in computer science from Chungnam National University, Daejeon, Rep. of Korea, in 2011. He is currently working in the Intelligent Network Research Team at the Electronics and Telecommunications Research Institute. He is mainly interested in intelligent edge computing and autonomous networks.

Sungback Hong received his BS degree in electronic telecommunication engineering from Kwangwoon University, Seoul, Rep. of Korea, and his MS degree in electronic engineering from Yonsei University, Seoul, Rep. of Korea, in 1982 and 1990, respectively. He received his PhD degree from Chungbuk National University, Cheongju, Rep. of Korea, in 2008. Since 1982, he has been with the Electronics and Telecommunications Research Institute (ETRI) and is currently with the Network Research Department as a principal researcher. His main research interests include edge computing, edge native applications, networking by artificial intelligence, networks for artificial intelligence, and low-latency networks for 6G.

Taeyeon Kim received his BS and MS degree in computer engineering from the Chung-Ang University, Seoul, Rep. of Korea, in 1990 and 1992 respectively, and his PhD in Network Engineering from the Chungbuk National University, Cheongju, Rep. of Korea, in 2008. He is currently leading the Network Intelligence Research Section at the ETRI, Rep. of Korea. He has been working in the area of information-centric networking and cloud networking with network function virtualization/software-defined network technologies since 1992. His interests include future networks, programmable networking infrastructure, network automation, and the Internet of Things with edge networking.