A Case for Two-stage Inference with Knowledge Caching

Geonha Park
KAIST
Daejeon, Republic of Korea
geonha@kaist.ac.kr

Changho Hwang
KAIST
Daejeon, Republic of Korea
chhwang@kaist.ac.kr

KyoungSoo Park
KAIST
Daejeon, Republic of Korea
kyoungsoo@kaist.ac.kr

ABSTRACT

Real-world intelligent services employing deep learning technology typically take a two-tier system architecture – a dumb front-end device and smart back-end cloud servers. The front-end device simply forwards a human query while the back-end servers run a complex deep model to resolve the query and respond to the front-end device. While simple and effective, the current architecture not only increases the load at servers but also runs the risk of harming user privacy.

In this paper, we present knowledge caching, which exploits the front-end device as a smart cache of a generalized deep model. The cache locally resolves a subset of popular or privacy-sensitive queries while it forwards the rest of them to back-end cloud servers. We discuss the feasibility of knowledge caching as well as technical challenges around deep model specialization and compression. We show our prototype two-stage inference system that populates a front-end cache with 10 voice commands out of 35 commands. We demonstrate that our specialization and compression techniques reduce the cached model size by 17.4x from the original model with 1.8x improvement on the inference accuracy.

CCS CONCEPTS

• **Computing methodologies** → **Mobile agents**; **Neural networks**; *Speech recognition*; • **Computer systems organization** → *n-tier architectures*; *Embedded software*; • **Security and privacy** → Privacy protections.

KEYWORDS

Neural networks; Embedded systems; Caching systems;

ACM Reference Format:
Geonha Park, Changho Hwang, and KyoungSoo Park. 2019. A Case for Two-stage Inference with Knowledge Caching. In *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications (EMDL'19), June 21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3325413.3329789

1 INTRODUCTION

Recent breakthroughs in deep learning are rapidly changing the lifestyle of modern homes. People now listen to music, check today's weather, or even set up an alarm through smart speakers [2, 6]. Recent smart refrigerators [8, 9] allow coordinating the schedule of family members or sending a reminder of expiration of stocked food. The demand for smart services is growing fast as the smart home market is expected to reach $107 billion by 2023 with compound annual growth rate (CAGR) of 9.5% [1].

Most of modern voice-based intelligent services take the centralized system architecture. A typical voice assistant wakes up with a special activation command [1], and passes up voice queries directly to a back-end cloud server. The back-end cloud server runs a complex deep model that recognizes a request, processes it, and sends a response back to the front-end device. This centralized query resolution is advantageous in that it not only makes the front device simple but also supports transparent enhancement of the service as the back-end deep model gets mature over time.

Unfortunately, the centralized scheme also brings significant drawbacks. First, even locally-executable queries that are popular must be forwarded to back-end servers. This increases the load at the back-end servers unnecessarily while it might impair the interactivity with a latency blowup. Second, some queries carry privacy-sensitive information that a user does not want to leak. Most queries inevitably hold some private information due to personal usage. Some people may be OK with mildly-sensitive information such as switching of TV channels, but few people may want to leak privacy-critical information such as a specific type of medicine kept in a smart refrigerator. Here, we ask a simple question. Why not execute popular or privacy-sensitive queries locally on a front-end device? [2]

The solution to this problem is widely known in computer systems design, i.e., caching. A simple, static cache with pre-loaded content might effectively address the problem. However, the central challenge lies in how we apply caching to a complex deep learning model. We refer to caching some features of a large deep model as *knowledge caching*. More formally, given a large deep model that handles a full query set of S with an average inference accuracy at k, how can we build a specialized deep model that processes only a subset of queries, Y ($Y \subset S$) with similar accuracy while it fits the resource budget of a front-end device?

Answering the question turns out to be non-trivial as it requires handling a number of challenges. The primary issue lies in the construction of an effective *deep model (DM) cache*. To build a DM cache, one might apply model specialization [11, 19, 21], a popular technique that trains a model only on a subset of tasks. However, model specialization raises two key problems: (a) given a query, how does it determine a cache hit or a miss and (b) how do we adjust the model size to run on a front-end device? The former focuses

[1]Like "Ok Google" or "Hello Alexa".
[2]Determining which queries should be executed locally is an open problem that we leave as future work.

on the correctness of the DM cache – it must tell what query it can answer, and what not, to avoid a wrong answer. In contrast, the latter concerns the efficiency as specialization itself does not reduce the model size. Second, a DM cache must be refreshed periodically to reflect the change in query popularity or to support individual needs for personalization. The key issue is how we minimize the re-training cost for building a new DM cache per front-end device. Third, we need to come up with a general strategy for building a DM cache. It would be nice to accommodate a variety of deep models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

In this paper, we discuss the challenges in knowledge caching, and provide our own approach to handling them. In addition, we present a prototype of a two-stage inference system that employs a DM cache at a front-end device. Our preliminary evaluation shows that a DM cache handling 28.6% of speech commands of a deep model [10] would require a specialized model that is 17.4x smaller than the original model while it improves the inference accuracy by 1.8x. We find that the DM cache decreases the inference time significantly by reducing the floating point operations by 17.3x.

2 OPPORTUNITIES & CHALLENGES

In this section, we discuss the opportunities for knowledge caching and practical challenges down the road.

2.1 Opportunities

Without accurate statistics on real-world queries, it is difficult to gauge the effectiveness of knowledge caching. Nevertheless, we discuss the potential for handling popular or privacy-sensitive queries at front-end devices.

Popular queries. According to a survey in 2018 [18], over 20% of the population in the U.S. already own a smart speaker while the average number of daily queries is 2.8 times. Not surprisingly, almost 80% of the queries are focused on top-7 categories [3] despite over 70k skills offered by a popular smart speaker [17]. Handling some of the popular query types on a front-end device would significantly reduce the load at cloud servers while it improves the user experience.

Private queries. According to the survey by PwC in 2018 [24], about 28% of the respondents that do not own a voice assistant express some concern about privacy. Given that voice assistants are usually placed in private locations at homes like living rooms (45.9%), kitchens (41.4%) or bedrooms (36.8%) [18] [4], one might worry that they may overhear private conversation [12]. Careful usage may mitigate the concern, but queries like "wake me up in 30 minutes" or "remind me of my schedule today" inevitably deal with private information. If knowledge caching is employed, the front-end device may be able to present an option to locally execute a private query or not to forward it to cloud servers unless it is granted by the user.

Computing capacity at front-end devices. A front-end device must have enough compute capacity for executing a DM cache locally. Fortunately, the recent trend shows that front-end devices are

[3]Such as checking the weather, setting a timer/an alarm, controlling smart home devices, listening to radio, asking for music, asking general questions, etc.

[4]The survey allows multiple responses.

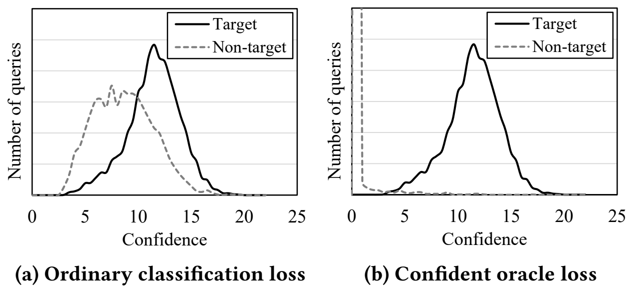

Figure 1: Confidence distributions of DM cache responses for target and non-target queries for training ResNet-56 with two different loss functions for the CIFAR-10 dataset. We use the Kullback-Leibler divergence of the output and uniform vectors as the confidence of a response.

getting more powerful. Smartphones such as Samsung's Galaxy S10 Plus [5] or Apple's iPhone X [4] employ high-performance processors with tens of GBs of memory, often with a dedicated neural network processors (NPUs) capable of handling 600 billion operations per second [4]. Other platforms like NVIDIA Jetson [7] or Google's AIY Edge TPU Boards [3] have an integrated CPU/GPU package with a few GBs of memory, achieving 750.1 and 32 GFLOPS, respectively. Given that a popular CNN model like ResNet-50 [15] requires 3.8 GFLOPs for inference, modern front-end platforms have enough capacity to offload some of machine learning tasks.

2.2 Challenges

Caching a subset of features of a complex deep model is a conceptually simple idea, but it presents a number of practical challenges. The primary difficulty is that the object for caching is stochastic computation rather than deterministic data accessed by a well-defined key, typically seen in other caching systems. In this section, we briefly summarize the issues and present our approach to tackling them.

(a) Determining a cache miss. One reasonable approach for a DM cache is to construct a specialized deep model trained for a specific target subset of the entire training dataset. Fortunately, specialized learning tends to enhance algorithmic performance (e.g. accuracy) [11, 19, 21] as it reduces the search space of a training algorithm. This property helps the DM cache to achieve high accuracy even when we compress the model as in Section 2.2-(c). However, one critical challenge is that a typical specialized model is unable to tell non-target queries from target queries. This implies that a DM cache cannot determine a cache miss even for a query that has not been learned.

Consider a classification task with n classes, where the output to a query is provided as a vector of scores for each class, $\mathbf{p} = \{p_0, p_1, \cdots, p_{n-1}\}$. Given a query, the class with the largest score in $\mathbf{p}$ is presented as the response. The $confidence$ of a response refers to the level of assurance that the model provides, and it is typically measured by the entropy [27] or Kullback-Leibler divergence [21, 27] of the output vector. Higher confidence implies that the model thinks its response is more likely to be correct. So, for a non-target query, the model must express low confidence in the

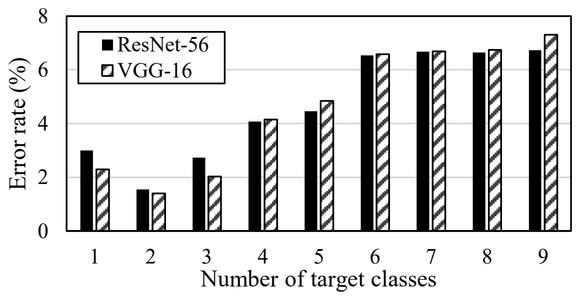

Figure 2: Error rates over varying numbers of target classes per model. We use the ResNet-56 and VGG-16 models with the CIFAR-10 dataset. Note that the error rate of a model with one target class is exceptionally high due to overfitting.

# Target classes	3	5	7	9
Error rate (%)	5.02	6.46	5.36	6.02
Max compression rate (%)	88	83	64	64

Table 1: Comparison of error rates and maximum compression rates over varying numbers of target classes per unit model. VGG-16 model and CIFAR-10 dataset are used. Error rate of the general VGG-16 model is 6.60%. In case of three target classes, the compression rate reached a theoretical limit, so we could not reduce the model size further.

response, where values of the output vector should be more or less uniform. Unfortunately, a typical specialized model produces responses with high confidence even for non-target queries. Figure 1a compares the distributions of confidence (measured in Kullback-Leibler divergence [5]) of target and non-target queries with the ResNet-56 model [15] when we specialize it with 5 classes out of 10 total classes. It shows that even the responses to non-target queries have high confidence similarly to those of target queries.

Our approach. We draw some insight from prior machine learning literature that tackles the similar issue in different contexts, such as regularization [14, 27], ensemble learning [21], and prevention of adversarial attacks [22]. While prior works focus on enhancing the accuracy and reducing the uncertainty of a model via handling confidence, we leverage it to resolve a cache miss with a DM cache. Specifically, we apply *confident oracle loss* [21] to a DM cache so that it trains the model to minimize the confidence for non-target queries. More specifically, at each training iteration with a randomly given training sample, we minimize ordinary classification loss (e.g. cross-entropy loss) if the sample is from a target class, and if not, we minimize $D_{\mathrm{KL}}(\mathrm{unif}\{0, n-1\} \,||\, \mathbf{p})$, where n is the number of target classes and D_{KL} is the Kullback-Leibler divergence, which measures the confidence of $\mathbf{p}$. Figure 1b shows that training with confident oracle loss effectively reduces the confidence on non-target queries. Note that training with confident oracle loss does not affect the accuracy of the model for target queries, compared with ordinary classification loss. This is because minimizing D_{KL} is stochastically equivalent to minimizing the classification loss of a randomly-picked non-target sample. In fact, the test accuracy of Figure 1a and Figure 1b is 4.58% and 4.56%, respectively.

(b) Avoiding model re-training. Cache eviction and replacement in conventional caching systems is simple as they deal with fixed data or deterministic computation for a cache item. However, cache replacement of a DM cache (due to popularity change or for personalization) requires re-training of a deep model with a different set of target queries. Model re-training not only incurs a long delay but it also consumes considerable computation resources. Front-end devices typically do not have enough resources for re-training nor have convenient access to a large amount of training dataset. Frequent re-training at back-end servers hardly scales as the number of front-end devices increases (i.e. # of models for training increases).

Our approach. We consider a simple solution that requires only one-time training cost. The idea is to leverage an ensemble of multiple unit models. First, we split n total classes into m ($m \leq n$) class groups so that each group has a set of co-related classes likely to be required as a whole. Then, we train a specialized model for each class group, which results in m unit models. We install the smallest set of unit models that collectively cover the classes locally handled on each front-end device. When a real query arrives, the front-end device builds a response by averaging the scores from individual unit models. If the confidence is uniformly small across the unit models, the DM cache considers the query as a cache miss as it is unlikely to be learned.

This approach raises one open issue – how do we compose class groups? If each group has many classes, the number of unit models for a DM cache could be made small. However, learning too many classes per model may degrade the inference accuracy. For instance, Figure 2 shows that a larger number of classes per unit model tends to increase the error rate. In practice, the accuracy is affected by the number of classes as well as the type of those a model learns. So, a prudent strategy should leverage both prior knowledge and careful analysis on the dataset. For simplicity, we assume that all classes are independent of each other in this work, and focus only on the number of classes to learn.

(c) Tailoring models to small devices. For practical deployment, the size of a DM cache must fit the resource budget of a front-end device while maintaining the inference accuracy comparable to that of the original model. Having a small cache is important as the size determines both the memory footprint as well as the inference latency of a query. Unfortunately, employing an ensemble of m unit models as a DM cache would end up with a size blowup by m times. A compression scheme like filter pruning [23] may reduce the model size, but its benefit is often not enough for a general model – the size reduction without loss of accuracy is limited to only 13.7% when we apply filter pruning to the ResNet-56 model for the CIFAR-10 dataset without specialization.

Our approach. Interestingly, we observe that a specialized unit model tends to compress much better with little degradation of accuracy. Table 1 compares the maximum compression rates of a unit model over varying numbers of target classes while keeping a comparable error rate to the general model (i.e. # of target classes is 10). All experiments use the VGG-16 model with the CIFAR-10

[5]A larger value represents higher confidence with 0 as the minimum.

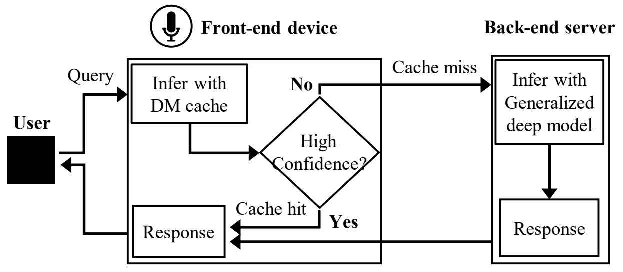

Figure 3: Two-stage inference architecture with a DM cache

dataset, and each model is compressed via filter pruning after specialized training of a unit model. The results show that a unit model specialized on a smaller number of classes compresses much better. Also, beyond a certain number of classes, the compression rate largely plateaus.

Based on the observation, we apply filter pruning to specialized unit models while keeping the number of classes per model small. After filter pruning, we optionally reduce the size of floating point parameters from 32 bits to 16 bits, which achieves extra compression at a small accuracy loss. Note that determining the number of classes per model is closely related to the open issue of composing class groups. If the value is too small, it would require training a large number of unit models, and each DM cache would need more unit models for an ensemble. If it is too large, the unit model may not compress well while the ensemble of a DM cache may represent classes not directly relevant to the cache. For now, we leave this problem as our future work.

Generality and limitation of our approach. Our suggestions so far (except model compression) are applicable to any supervised learning tasks based on deep neural networks. We have verified that our model specialization and ensemble inference work well with various CNN-based tasks such as image classification [15, 30–32], object detection [29], and speech commands recognition [10], and a RNN-based task that recognizes hand-written Chinese characters [34]. Other learning tasks such as unsupervised learning [13, 28] or reinforcement learning [25] would require a different specialization technique as our current approach assumes class labeling to tell target from non-target tasks. For model compression, we have tried only filter pruning [23], generally applicable to CNN models. We leave it as our future work to try out other compression methods [26] to support neural networks beyond CNNs.

3 SYSTEM OVERVIEW

We build a simple prototype system that performs two-stage inference with a query. Figure 3 shows the overall architecture of our system. The front-end device receives a query from a user and feeds it to its DM cache consisting of multiple unit models. If the response of the cache has higher confidence than a pre-defined threshold (i.e. cache hit), the device executes the function associated with the response (e.g., turning on TV after inferring the voice query, "Turn on TV") and delivers the result to the user. In case of a cache miss, the front-end device forwards the query to a back-end server, and relays the result from a general deep model at the server to the user.

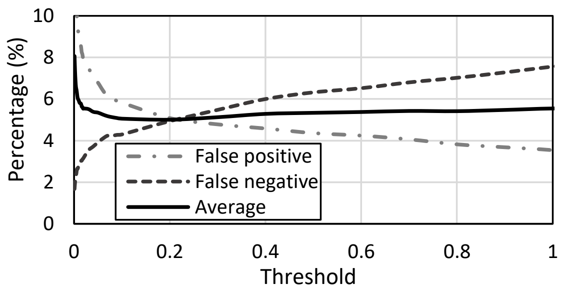

Figure 4: False positive and negative rates over varying confidence thresholds. ResNet-56 model specialized on 5 classes out of 10 total classes of the CIFAR-10 dataset. Validation data consists of target and non-target classes of the same size.

In this section, we present the process of building the two-stage inference system step by step.

3.1 Offline Model Building

The first step for building the two-stage inference system is to train a general deep model for back-end servers. Then, we train a set of specialized unit models used as a building block for a particular DM cache. So, we mainly focus on building the unit models here.

Arranging class groups. The admin needs to determine the number of class groups, m, considering co-related tasks and the model size. Then, each unit model would have roughly $\lceil n/m \rceil$ target classes to specialize on. Determining the optimal value of m is tricky as a large value would increase the number of unit models per DM cache, which may exceed the memory footprint budget. On the other hand, too small a value of m may increase the error rate of each unit model. Our current strategy is to start with a reasonable number of classes (e.g., 3-4) per unit model and allow some overlap of classes over unit models. Then, we build a DM cache as an ensemble of unit models and evaluate the accuracy and model size. If it does not work out, then, we repeat the process with a different number of per-model classes.

Training unit models. For specialized training of a unit model, we start with a general deep model structure but train it with confidence oracle loss. Then, we compress the trained model via filter pruning, and fine-tune the model with distillation learning [16], which helps improve the accuracy. We repeat this procedure until we find the largest compression rate that produces comparable accuracy to the general deep model. We employ binary search to minimize the steps to find the optimal compression rate.

Fine-tuning via distillation learning. Model compression tends to remove parameters with small values that are less likely to affect overall computation results. However, it may also remove important parameters that influence the final accuracy. This behavior can be fixed via some extra training after compression called fine-tuning. For fine-tuning in our system, we apply distillation learning [16] with the general model. To goal of distillation learning is to inherit the generalizability of a larger model to a smaller model, which often enhances the accuracy of the latter. To apply it in the context of specialized training, we slightly modify the training procedure.

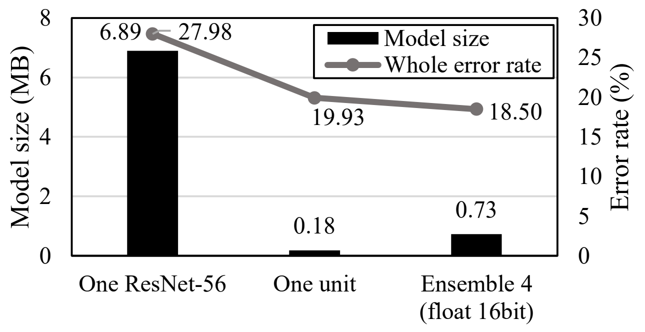

Figure 5: Comparison of model sizes and error rates of a general model (ResNet-56) vs. a DM cache (an ensemble of 4 unit models).

If a training sample is from a target class, we first infer the sample with the general model. If the response is correct, we minimize classification loss of the unit model for both the ground truth label and the score vector returned from the general model. This means that the unit model learns not only the ground truth but also how the general model responds, only when the general model returns a correct response. Otherwise, the training works the same as described before in Section 2.2-(a).

Determining confidence threshold. As shown in Figure 4, a higher confidence threshold reduces a false positive rate (i.e. wrong cache hits) while it increases a false negative rate (i.e. wrong cache misses). For balanced performance, we choose the confidence threshold that minimizes the average of false positive and negative rates. However, one can use a different value depending on the learning task. For example, one can use a lower threshold to favor local execution if the task is tolerant on occasional wrong responses.

3.2 Online Adjustment

Query preference of a user may change over time, which would increase cache misses and reduces the effectiveness of the DM cache. To detect query popularity change, the back-end servers continuously monitor user queries by analyzing cache misses as well as periodic reports from the front-end device. When cache replacement is desirable, the back-end server builds and deploys an ensemble of unit models that cover the new set of popular classes.

4 EXPERIMENTS WITH A DM CACHE

We present preliminary evaluations of our prototype system with two workloads.

Feasibility testing. To test the feasibility of our system, we first run an image classification benchmark with the ResNet-56 model [15] for the CIFAR-100 dataset [20].

Out of 100 total classes, we build a DM cache that handles 10 classes at a front-end device. First, we split 100 classes into 40 class groups where each group specializes on 3 classes. Having an overlap in learned classes over unit models tends to improve the accuracy, so we arrange some classes (e.g., 20 classes) to be overlapped across multiple unit models. To handle 10 classes at the front-end device, we build an ensemble of 4 unit models, following the approach described earlier. Beyond filter pruning, we also reduce the size of a floating point parameter from 32 bits to 16 bits. For experiments, we feed the input queries of target and non-target classes with an equal weight. For a target class query, we count the answer as correct if the cache determines the query as cache hit and produces a correct label for it. For a non-target class query, we count it as correct only if the cache flags the query as cache miss.

Figure 5 compares the error rates and normalized model sizes. Interestingly, the DM cache shows a 1.5x better error rate with the aggregate model size 9.4x smaller than the general model. The reasons for higher accuracy are two folds. First, the compression rate already reaches a theoretical limit, so we could not reduce the model size further. This means that the accuracy improvement is achieved at the cost of a slightly larger model (9.4x instead of 10x size reduction). Second, we find that the ensemble of unit models improves the accuracy beyond what individual models achieve.

Model type	Size (MB)	MFLOPs	Error (%)
General	80.1	33.3	9.62
General (compressed)	26.1	10.9	13.63
DM cache (w/o DL)	4.6	1.9	5.93
DM cache (w/ DL)	4.6	1.9	5.57

Table 2: Comparison of the general model with the DM cache for speech commands recognition. DL = Distillation learning. MFLOPs = Million floating point operations.

Speech commands recognition. We gauge the effectiveness of our system with small-scale speech recognition. We implement a CNN model for speech commands recognition of simple keywords by referring to the implementation of TensorFlow tutorial [10]. We use a small speech commands dataset [33] consisting of 35 different keywords, and build a DM cache that handles only 10 speech commands out of them. Our scenario is that these 10 commands are locally executable on a front-end device while the rest of 25 commands should be forwarded to the back-end cloud servers.

Table 2 shows that our target-aware specialization and filter pruning effectively reduce the model size by 17.4x and 5.7x from the original and compressed models, achieving higher accuracy in classification. We suspect that the higher accuracy stems from the fact that the DM cache regularizes the model better. We find distillation learning brings extra improvement of accuracy by 0.36%.

5 CONCLUSION

In this paper, we have presented the feasibility of knowledge caching as a two-stage model inference system. To the best of our knowledge, this work presents the concept of caching a select features of an arbitrary deep model for the first time. We find that a deep model cache at a front-end device is often desirable for executing popular or private operations, and we have discussed a number of challenges in building an effective deep model cache. We believe target-aware model specialization as well as an ensemble of compressed unit models would be a reasonable first step towards constructing a model cache.

ACKNOWLEDGMENTS

We appreciate the feedback by anonymous reviewers of EMDL'19, and thank Celso Moraes for early discussion on knowledge caching. This work is in part supported by the ICT Research and Development Program of MSIP/IITP, Korea, under projects, [2016-0-00563, Research on Adaptive Machine Learning Technology Development for Intelligent Autonomous Digital Companion] and [2015-0-00590, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development].

REFERENCES

[1] 2018. *Smart Home Market Report: Trends, Forecast and Competitive Analysis.* Technical Report. Lucintel, 8951 Cypress Waters Blvd., Suite 160, Dallas.

[2] 2019. Echo & Alexa - Amazon Devices. Retrieved April 10, 2019 from https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?node=9818047011

[3] 2019. Edge TPU – Run Inference at the Edge. Retrieved April 10, 2019 from https://cloud.google.com/edge-tpu/

[4] 2019. The future is here: iPhone X. Retrieved April 10, 2019 from https://www.apple.com/newsroom/2017/09/the-future-is-here-iphone-x/

[5] 2019. Galaxy S10 Performance. Retrieved April 10, 2019 from https://www.samsung.com/us/mobile/galaxy-s10/performance/

[6] 2019. Google Home. Retrieved April 10, 2019 from https://store.google.com/product/google_home

[7] 2019. High Performance AI at the Edge | NVIDIA Jetson TX2. Retrieved April 10, 2019 from https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/

[8] 2019. LG Smart Refrigerators: Powered by SmartThinQ IOT. Retrieved April 10, 2019 from https://www.lg.com/us/discover/smartthinq/refrigerators

[9] 2019. Samsung Family Hub Smart Refrigerator. Retrieved April 10, 2019 from https://www.samsung.com/us/explore/family-hub-refrigerator/refrigerator/

[10] 2019. TensorFlow audio recognition tutorial. Retrieved April 10, 2019 from https://www.tensorflow.org/tutorials/sequences/audio_recognition

[11] Grigory Antipov, Moez Baccouche, Sid-Ahmed Berrani, and Jean-Luc Dugelay. 2016. Apparent Age Estimation from Face Images Combining General and Children-Specialized Deep Learning Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[12] Katie Canales. 2018. A couple says that Amazon's Alexa recorded a private conversation and randomly sent it to a friend. Retrieved April 10, 2019 from https://www.businessinsider.com/amazon-alexa-records-private-conversation-2018-5

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NIPS)*.

[14] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *International Conference on Machine Learning (ICML)*.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[16] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531.

[17] Bret Kinsella. 2018. There are Now More Than 70,000 Alexa Skills Worldwide, Amazon Announces 25 Top Skills of 2018. Retrieved April 10, 2019 from https://bit.ly/2VLyqJ9

[18] Bret Kinsella and Ava Mutchler. 2018. Smart Speaker Consumer Adoption Report. Retrieved April 10, 2019 from https://voicebot.ai/wp-content/uploads/2018/10/voicebot-smart-speaker-consumer-adoption-report.pdf

[19] Morten Kolbæk, Zheng-Hua Tan, and Jesper Jensen. 2017. Speech Intelligibility Potential of General and Specialized Deep Neural Network Based Speech Enhancement Systems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25, 1, 149–163.

[20] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images.* Master's thesis. Department of Computer Science, University of Toronto.

[21] Kimin Lee, Changho Hwang, KyoungSoo Park, and Jinwoo Shin. 2017. Confident Multiple Choice Learning. In *International Conference on Machine Learning (ICML)*.

[22] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. 2018. A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks. In *Advances in Neural Information Processing Systems (NIPS)*.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations (ICLR)*.

[24] Mark McCaffrey, Paige Hayes, Jason Wagner, and Matt Hobbs. 2018. *Consumer Intelligence Series: Prepare for the voice revolution.* Technical Report.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602.

[26] Sharan Narang, Greg Diamos, Shubho Sengupta, and Erich Elsen. 2017. Exploring Sparsity in Recurrent Neural Networks. In *International Conference on Learning Representations (ICLR)*.

[27] Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton. 2017. Regularizing Neural Networks by Penalizing Confident Output Distributions. In *International Conference on Learning Representations (ICLR)*.

[28] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*.

[29] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *CoRR* abs/1804.02767.

[30] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[31] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[33] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR* abs/1804.03209.

[34] Xu-Yao Zhang, Fei Yin, Yan-Ming Zhang, Cheng-Lin Liu, and Yoshua Bengio. 2018. Drawing and Recognizing Chinese Characters with Recurrent Neural Network. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 40, 4, 849–862.