2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS) | 978-1-6654-4935-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/MASS52906.2021.00013

Adaptive Deep Neural Network Ensemble for Inference-as-a-Service on Edge Computing Platforms

Yang Bai†, Lixing Chen‡, Letian Zhang†, Mohamed Abdel-Mottaleb†, *Fellow, IEEE,* Jie Xu†, *Senior Member, IEEE,*
†Department of Electrical and Computer Engineering, University of Miami, FL, USA
‡Institute of Cyber Science and Technology, Shanghai Jiao Tong University, China

***Abstract*—The momentous enabling of deep learning (DL)-powered mobile application is posing a soaring demand for computing resources that can hardly be satisfied by mobile devices. In this paper, we employ *Edge Computing* to deliver DL inference services to mobile users, where Deep Neural Networks (DNNs) are configured on edge servers, processing inference tasks received from mobile devices. A novel method called Adaptive DNN Ensemble (ADE) is proposed to enhance the performance of DL inference services. The core of ADE is the *DNN ensemble technique* which improves the stability and accuracy of DL inference. Due to the limited computing resources and service response deadline, ADE needs to judiciously determine DNNs to be included in the DNN ensemble, which poses a unique DNN ensemble selection problem. In addition, because DNNs exhibit performance variations for tasks with different features, DNN ensemble selection also aims to reconfigure DNN ensembles according to the feature of admitted tasks. We design an online learning algorithm, *Contextual Combinatorial Multi-Armed Bandit* (CC-MAB), to learn the DNN performance for tasks with different features. We rigorously prove that the proposed online learning algorithm is able to achieve asymptotic optimality. Experiments are carried out on an edge computing testbed to evaluate our method. Various implementation concerns, including memory usage, time complexity, and DNN switching cost, are considered. The results show that ADE outperforms other benchmarks in terms of inference accuracy and can provide real-time responses.**

I. Introduction

Smartphones and hand-held devices are now closely tied to deep learning (DL) intelligence in many of their functionalities, including speech-based assistants (e.g., Siri, Cortana) and face recognition enabled phone-unlock (e.g., FaceID). This trend is continuously driving the advance of deep learning techniques for mobile devices. New-generation hardware, e.g., Apple neural engine [1], is designed to accelerate neural network processing. Lightweight deep learning libraries (e.g., Tensorflow Lite [2] and Core ML [3]) are built to support DL applications on mobile devices. Novel deep learning techniques, e.g. Deep Neural Network (DNN) compression [4], [5] and knowledge distillation [6], help compress large-scale DNN models into compact models that fit the size of on-chip RAM. While these techniques enable mobile devices to run DNNs, they are unlikely to be universal solutions due to the substantial heterogeneity of mobile devices in terms of their computing capacities. A recent study by Facebook [7] shows that over 50% mobile devices are using processors at least six years old, limiting what is possible of DL services. Besides, Running DL inferences frequently also drains the battery fast [8], [9]. Therefore, external boosters become necessary to realize the full potential of DL intelligence for mobile devices. The recently emerged *Edge Computing* [10] is envisioned to be a promising alternative for supporting mobile DL intelligence [11], [12]. Being physically close to users and leveraging fast network technologies such as 5G, edge computing promises several benefits compared to the traditional cloud-based computing paradigm, including lower latency, higher energy efficiency, better privacy protection, and reduced bandwidth consumption [13]. With the assistance of computation offloading techniques [14], the edge computing platform becomes an optimal site for providing DL services.

This paper proposes a novel mechanism to enhance the performance of DL service on edge computing platforms. The core of our mechanism is using DNN ensemble techniques to improve the stability and accuracy of DL inference. The DNN ensemble technique has been providing state-of-the-art performances for many learning problems. For example, the winning teams of ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) in the latest four consecutive years all incorporate the DNN ensemble technique in their method [15]. Despite this advantage, using the DNN ensemble is originally unfavorable to mobile applications because running multiple DNNs requires a large amount of computing resources that cannot be satisfied by resource-constrained mobile devices. The deployment of edge computing platforms provides mobile users the access to sufficient computing resources, making it possible for trading computing resources for better inference performances with the DNN ensemble technique.

This work is supported in part by NSF under grants 2006630, 2033681, 2029858 and 2044991.

Y. Bai, L. Zhang, M. Abdel-Mottaleb, J. Xu are with the Department of Electrical and Computer Engineering, University of Miami, FL, 33146. E-mail: y.bai9@umiami.edu, {lxz437, mottaleb, jiexu}@miami.edu.

L. Chen is with the Institute of Cyber Science and Technology, Shanghai Jiao Tong University, China, 200240. E-mail: lxchen@sjtu.edu.cn

978-1-6654-4935-9/21/$31.00 ©2021 IEEE
DOI 10.1109/MASS52906.2021.00013

However, there exist several challenges to bring the DNN ensemble technique to edge computing platforms. **1)** The first challenge is the limited computing resource at edge servers. Although computing resources on the edge platform is much powerful than mobile devices, it is still limited compared to Cloud [16]. Therefore, running complicated DNN ensembles is still prohibitive, and may incur large inference delay that is unfriendly to latency-critical edge services. This requires the operator to judiciously decide *how many* and *which* DNNs to include in DNN ensembles, namely the *DNN ensemble selection problem*. **2)** The second challenge is the DNN heterogeneity. The DNNs collected by a service provider may come from different sources. This is a natural assumption because the data collected by a single institution for training DNNs is often limited. While data sharing is desirable for application service developers to obtain high-quality DNNs, it oftentimes leads to severe privacy issues, especially for clinical, financial, social data. The trained DNN is a generalization of structured knowledge that contains less privacy-sensitive information, and therefore sharing DNNs is more welcomed than directly sharing the source data. For example, Facebook has disclosed the open-source release of its Deep Learning Recommendation Model (DLRM) [17] but veils the source data due to privacy concerns. Therefore, DNNs collected by a service provider can be trained/validated on *different* data sources with *different* DNN architectures by *different* institutions, and hence achieving *different* performances. These four "*different*"s characterize the heterogeneity of DNNs and further justify the necessity of DNN ensemble selection. **3)** The third challenge is the unknown in-use performance of DNNs. While the DNN performance can be evaluated on standard test data, one cannot guarantee that the users' input data comes from the same distribution as the standard test data. The actual performance delivered by DNNs is only revealed during implementation and needs to be learned over time. **4)** The fourth challenge is the variability of user tasks. The features of inference tasks vary due to many external factors. For example, consider the image as DNN input, the device camera determines the noise and resolution of captured images, and the time and location may affect the brightness of images. Different DNNs usually have different sensitivity to these factors and hence their inference qualities also vary across tasks with different features. In other words, there is no "master key" for all inference tasks and the discrimination of the "right key" for certain inference tasks is crucial. This requires the service provider to adaptively configure the DNN ensemble with the best-fit DNNs for admitted tasks.

This paper presents an Inference-as-a-Service framework on edge computing platforms to deliver DL inference services to mobile users. The proposed framework utilizes DNN ensemble techniques and aims to address the DNN ensemble selection problem to improve the efficiency and quality of DL inference. The key contributions are summarized as follows:

1) An Inference-as-a-Service framework is designed for edge computing platforms. The proposed framework utilizes DNN ensemble techniques for enhancing the DL inference quality. A unique problem, DNN ensemble selection, is investigated due to the limited edge computing resource. The goal DNN ensemble selection is to adaptively configure the DNN ensemble with DNNs that works the best for currently admitted tasks.

2) We propose a method called Adaptive DNN Ensemble (ADE) for solving the DNN ensemble selection problem. ADE utilizes a novel multi-armed bandit algorithm called *contextual combinatorial multi-armed bandit* (CC-MAB). It learns in-use performances of DNNs and recruits DNNs into DNN ensembles based on the features of admitted tasks. ADE judiciously balances exploration (i.e., learning DNN performance) and exploitation (i.e., optimizing inference performance of DNN ensembles based on the learned knowledge), and provides asymptotic optimality.

3) We build an edge computing testbed to evaluate the performance ADE. The experiment is performed on the WIDER-attribute dataset [18] with a variety of pre-trained DNNs. In particular, we optimize implementation schemes for running DNN ensembles on the edge server and reduce DNN switching costs incurred by reconfigurations of DNN ensembles. The experimental results show that ADE improves the inference accuracy by 11.3% over the most-capable DNN.

The rest of this paper is organized as follows: Section II gives the system model and defines the DNN ensemble selection problem. Section III designs our online learning algorithm. Section IV gives the experimental results, followed by conclusions in Section V.

II. System Model

A. Edge Computing Platform

Our Inference-as-a-Service framework is compatible with most edge computing systems. Let us consider a general multi-access edge computing (MEC) system [10], [19] with multiple edge sites, multiple mobile users, and multiple DL services. The MEC system operator manages computing resources on edge servers using virtualization techniques, e.g., virtual machines (VMs) and containers. To provide DL services on an edge site, the service provider requests computing resources (e.g., VMs) from the edge server using certain mechanisms, e.g, resource rental planning [20]. When the computing resource is allocated, the service provider configures its appli-

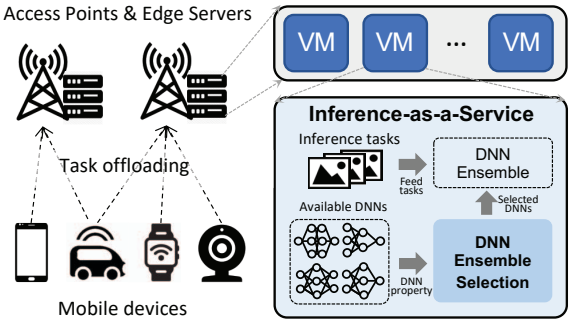

Fig. 1: Illustration of the system model.

cation on the edge server and users covered by the edge site can send DL inference tasks to the edge server. The focus of our work is enhancing the performance of DL inference services on the edge server, and this problem is independent for each DL service provider once the computing resource on the edge server is allocated. Without loss of generality, we present our method for one DL inference service on one edge server. Fig. 1 illustrates the considered system model.

The DL service provider collects a set of DNN models, indexed by $\mathcal{M} = \{1, 2, ..., M\}$, and stores them on the edge server. We assume that all available DNNs can be stored on the edge server since the storage is less likely to be a constraint nowadays due to its low price. The operational timeline for the service provider is discretized into time slots $t = 1, 2, \ldots, T$ (e.g., a few seconds per slot). In each time slot t, users offload inference tasks to the edge server. The task offloading decision is made by users and hence is assumed to be an independent process. We let $X^t = \{x_j^t\}_{j=1}^{N_t}$ be the inference tasks admitted by the edge server at the beginning of time slot t, where N^t is the total number of admitted inference tasks. The edge service provider then loads DNNs into the edge server RAM (or VRAM if GPU is used) to process the admitted tasks.

B. Inference with DNN Ensembles

Using the DNN ensemble technique, the service provider runs multiple DNNs on the edges server and fuses outputs of individual DNNs to generate a final inference result. While the DNN ensemble improves the inference quality, it also incurs higher resource usage and time complexity due to running multiple DNNs. Consider the limited computing capacity on the edge server and the potential requirements on the response time, it is not always possible to include all available DNNs in the DNN ensemble. Therefore, we limit the number of DNNs that can be included in the DNN ensemble, denoted by B. The value of B should be chosen to guarantee that running any B DNNs in $\mathcal{M}$ will not exceed the computing capacity of the edge server and will not incur inference delay larger than the response deadline. Let $\mathcal{S}^t \triangleq \{s_1^t, s_2^t, \dots\} \subseteq \mathcal{M}$ with $|\mathcal{S}^t| \leq B$ be the DNN ensemble selected in time slot t. The admitted inference tasks are forwarded to each DNN in $\mathcal{S}^t$ and the outputs of DNNs are combined by a fusion rule (discussed in the next subsection) to generate a final result.

C. Fusion Rule and Utility

The edge service provider derives utilities by providing better inference quality, e.g., prediction accuracy. Without loss of generality, this paper considers DL-based classification problem as a DL inference service. Note that our method can be easily extended to other DL services with very slight modification on the fusion processes. For classification problems, the output of a DNN $m \in \mathcal{M}$ for a task $x_j^t \in X^t$ is a vector of classification confidences $\boldsymbol{c}_m(x_j^t) = \{c_{m,k}(x_j^t)\}_{k=1}^K$, where K is the total number of classes and $c_{m,k}(x_j^t)$ denotes the confidence of class k being the true class for task x_j^t. DNN m gives the rank-1 classification result defined as $\hat{y}_{m,j}^t = \arg\max_k c_{m,k}(x_j^t)$. Let y_j^t be the ground truth of task x_j^t and then the classification correctness of DNN m for x_j^t is $\mathbf{1}\{\hat{y}_{m,j}^t = y_j^t\}$ where $\mathbf{1}\{\cdot\}$ is an indicator function. If the service provider only uses DNN m for inference, then the final result $\hat{y}_j^t$ for task x_j^t is $\hat{y}_j^t \leftarrow \hat{y}_{m,j}^t$. However, the service provider may run multiple DNNs with DNN ensemble techniques and hence a fusion rule is needed to combine outputs of multiple DNNs. We utilize weighted confidence as our fusion rule. We note that other fusion rules are also compatible with our method. The weighted confidence method jointly considers the confidence outputs $\boldsymbol{c}_m(x_j^t), \forall m \in \mathcal{S}^t$ of selected DNNs based on their accuracy $q_{m,j}^t \triangleq \Pr\{\hat{y}_{m,j}^t = y_j^t\}$ for task x_j^t. Although these accuracy values are unknown to the service provider a priori, we will design an online learning algorithm to learn this information (elaborated later in the paper). The weighted confidence of DNN ensemble for class k is calculated by:

$$\tilde{c}_k(x_j^t) = \frac{\sum_{m \in \mathcal{S}^t} q_{m,j}^t c_{m,k}(x_j^t)}{\sum_{m \in \mathcal{S}^t} q_{m,j}^t}.$$

The final inference result for x_j^t is determined by $\hat{y}_j^t = \arg\max_k \tilde{c}_k(x_j^t)$. We generalize the fusion process into a mapping function $\hat{y}_j^t = f(x_j^t, \mathcal{S}^t, \boldsymbol{q}_j^t)$ which maps task x_j^t, DNN ensemble $\mathcal{S}^t$, and DNNs' accuracy $\boldsymbol{q}_j^t \triangleq \{q_{m,j}^t\}_{m \in \mathcal{M}}$ to an inference result $\hat{y}_j^t$. The reward for task x_j^t is defined as the correctness of its inference result

$$u(x_j^t, \mathcal{S}^t, \boldsymbol{q}_j^t) = \mathbf{1}\{f(x_j^t, \mathcal{S}^t, \boldsymbol{q}_j^t) = y_j^t\} \tag{1}$$

Given the set of admitted tasks X^t in time slot t, the expected reward for the service provider in time slot t is

$$U(X^t, \mathcal{S}^t, \boldsymbol{q}^t) = \sum\nolimits_{x_j^t \in X^t} \mathbb{E}\left[u(x_j^t, \mathcal{S}^t, \boldsymbol{q}_j^t)\right] \tag{2}$$

where $\boldsymbol{q}^t = \{\boldsymbol{q}_j^t\}_{j=1}^{N^t}$ is DNNs' accuracy for all tasks in X^t.

D. Problem Formulation and Oracle Solution

The service provider aims to maximize its expected reward in a total of T time slots by selecting the best-fit DNN ensembles $\{\mathcal{S}^t\}_{t=1}^T$ in T time slots. The DNN ensemble selection problem is formally defined as:

$$\mathscr{P}1: \max_{\{\mathcal{S}^t\}_{t=1}^T} \sum\nolimits_{t=1}^T U(X^t, \mathcal{S}^t, \boldsymbol{q}^t) \tag{3a}$$

$$\text{s.t.} \quad \mathcal{S}^t \subseteq \mathcal{M}, |\mathcal{S}^t| \leq B, \forall t \tag{3b}$$

The main difficulty for solving the above problem is the unknown DNN accuracy $\boldsymbol{q}^t$ for user tasks. Therefore, $\mathscr{P}1$ is not merely a long-term optimization problem, but involves learning of DNN accuracy during the DNN ensemble selection. We assume for now the existence of an *oracle* that knows precisely the DNN accuracy for user tasks and give an oracle solution to $\mathscr{P}1$. With the oracle information, $\mathscr{P}1$ is fully decoupled for each time slot and can be divided into T independent subproblems, one for each time slot t as follows:

$$\mathscr{P}2: \max_{\mathcal{S}^t} \mathbb{E}\left[U(X^t, \mathcal{S}^t, \boldsymbol{q}^t)\right], \text{ s.t. } \mathcal{S}^t \subseteq \mathcal{M}, |\mathcal{S}^t| \leq B.$$

However, it is still difficult, if not impossible, to solve the per-slot problem $\mathscr{P}2$ optimally due to the obscured fusion

process. The performance of DNN ensembles depends on many factors, e.g., the orthogonality and complementary of DNNs' training/validation data [21], which are unknown and difficult to characterize analytically. Therefore, it is extremely, if not impossible, to give an analytical solution to $\mathscr{P}2$. Fortunately, previous studies [21], [22] have provided valuable empirical experience on how to pick a good DNN ensemble. Basically, we need to answer two questions: *"How many DNNs should be included in the ensemble?"* and *"What kind of DNNs are preferred to be included in the ensemble?"*.

1) Determining Ensemble Size: Earlier studies [23], [24] show that adding members to an ensemble has a diminishing return effect and the error reduction appears to reach a plateau after 25 members. More recent work [25] suggests that the ideal number of members in an ensemble should be similar to the number of classes in the classification problem. Further adding members into the ensemble may deteriorate the accuracy. Usually, the number of classes in image classification problems is much larger than the number of DNNs that can be loaded on the edge server. Therefore adding a DNN to an ensemble is always beneficial, which means that $|\mathcal{S}^t| = B, \forall t$.

2) Identify Best-fit DNNs: A variety of ensemble selection methods are investigated in the literature. For example, [26] selects classifiers according to their local accuracy, and [21] measures the diversity of classifiers and selects the most independent classifiers to construct ensembles. However, from the experiments of these works, it is observed that the results are strongly affected by the applied data. A recent work [22] compared the performance of a variety of ensemble selection methods. The results indicate that in most cases picking the classifiers with the highest accuracy to build the ensemble classifier will produce the best inference performance.

Heuristic Rule: Based on the above observations, the solution to $\mathscr{P}2$ becomes picking B DNNs that are expected to have the highest accuracy for the admitted tasks X^t, i.e.,

$$\max_{\mathcal{S}^t} \frac{1}{N^t} \sum\nolimits_{j=1}^{N^t} q_{m,j}^t, \text{ s.t. } \mathcal{S}^t \subseteq \mathcal{M}, |\mathcal{S}^t| \leq B. \quad (4)$$

Therefore, our DNN ensemble selection problem becomes learning the accuracy of collected DNNs. However, DNNs usually have different performance for tasks with different features. For example, in image classification problems, the resolution, brightness, hue, and contrast of images may affect the performance of DNNs. This should be considered when learning the DNN accuracy. Hereinafter, we call these features the *context* of inference tasks. In the following, we propose an online algorithm to learn the DNN accuracy based on the task context, and cast the DNN ensemble selection into a Contextual Combinatorial Multi-armed Bandit (MAB) problem [27].

III. Adaptive DNN Ensemble via Contextual Combinatorial MAB

A. Context-parameterized Accuracy

This paper considers simple context information, e.g. image resolution and contrast, that can be directly observed without processing inference tasks, and hence using the context does not incur extra computation burdens. The underlying assumption for using context is that a DNN will have similar performance for inference tasks with similar contexts. This assumption is natural and can be partially verified by the experiment in Section IV. This allows us to learn the DNN accuracy for a group of inference tasks with similar context, which significantly improves the learning efficiency. We let $\omega_j^t \in \Omega$ denote the context associated with task x_j^t, where Ω is the context space. We slightly abuse the notation of DNN accuracy $q_{m,j}^t$ by defining the context-parameterized accuracy $q_m(\omega_j^t)$, i.e., the accuracy of DNN m for task x_j^t depends on the context ω_j^t. We let $\mu_m(\omega) = \mathbb{E}[q_m(\omega)]$ be the expected accuracy of DNN m for inference tasks with the context ω. The oracle knows the expected accuracy $\mu_m(\omega)$ for an arbitrary context ω, and selects DNNs B DNNs that have the highest expected accuracy for admitted tasks:

$$s_b^{*t} \in \underset{m \in \mathcal{M} \setminus \cup_{i=1}^{b-1} s_i^{*t}}{\arg\max} \frac{1}{N^t} \sum\nolimits_{j=1}^{N^t} \mu_m(\omega_j^t), \ b = 1, \ldots, B. \quad (5)$$

We let $\mathcal{S}^{*,t} = \{s_1^{*,t}, \ldots, s_B^{*,t}\}$ denote the oracle DNN ensemble decision in time slot t. In practice, the service provider does not have a priori knowledge on the context-parameterized accuracy, and a learning algorithm is needed to learn this accuracy information. Next, we show our learning method, Adaptive DNN Ensemble (ADE).

B. CC-MAB for Adaptive DNN Ensemble

ADE is designed based on the framework of contextual combinatorial MAB (CC-MAB). The operations of ADE in each time slot t are as follows: (i) the service provider observes the context ω_j^t of each inference tasks $x_j^t \in X^t$ admitted by the edge server. (ii) A DNN ensemble $\mathcal{S}^t$ is selected based on the context of admitted tasks and the estimated DNN accuracy learned from previous time slots. (iii) The inference tasks are forwarded into each DNN in the selected DNN ensemble, and outputs of individual DNNs are fused to final inference results that are returned to users. (iv) At the end of the time slot, the ground-truths of inference tasks are observed and the estimated context-parameterized accuracy for each selected DNN is updated.

Obtaining precise estimations of context-parameterized accuracy for a DNN requires an adequate collection of inference results for tasks with different contexts. Note that the performance of a DNN is revealed only when it is selected and used in the DNN ensemble, and therefore the purpose of selecting a DNN can be either *exploration*, i.e., to learn the accuracy of DNN for tasks with a particular context, or *exploitation*, i.e., to select DNNs that are expected to deliver the highest accuracy. An important designing goal in MAB problems is balancing the tradeoff between exploration and exploitation.

The pseudocode of ADE is presented in Algorithm 1. ADE starts by partitioning the context space into small uniform hypercubes, i.e., creating groups of similar contexts. Specifically, a partition Λ_T is created on the context space Ω given the time horizon T. We consider a bounded context space that can be written as $\Omega \triangleq [0,1]^D$, where D is the dimension of

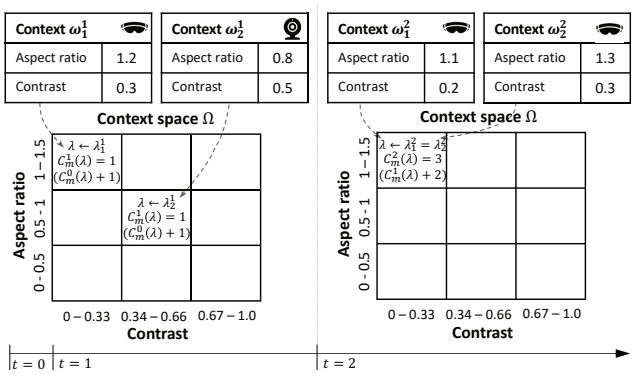

Fig. 2: Illustration of context partition and counter update. *We only show the update of counter $C_m^t(\lambda)$, the update for $E_m^t(\lambda)$ is similar.*

context space Ω. ADE splits the context space Ω into $(h_T)^D$ hypercubes λ, $\lambda \in \Lambda_T$ with identical size of $\frac{1}{h_T} \times \cdots \times \frac{1}{h_T}$. The parameter h_T is an important algorithm parameter to be designed for determining the number of hypercubes in the partition. For each task x_j^t, ADE determines hypercube $\lambda \in \Lambda_T$ that context ω_j^t belongs to, i.e., $\omega_j^t \in \lambda_j^t$ holds. Let $\boldsymbol{\lambda}^t = \{\lambda_j^t\}_{j=1}^{N^t}$ collect hypercubes for all admitted tasks. ADE keeps two counters, counter $C_m^t(\lambda)$ and counter $E_m^t(\lambda)$, for each pair of DNN $m \in \mathcal{M}$ and hypercube $\lambda \in \Lambda_T$. $C_m^t(\lambda)$ records the number of tasks with context in hypercube λ that have been processed by DNN m up to time slot t; and $E_m^t(\lambda)$ records the number of tasks with context in hypercube λ that are correctly classified by DNN m (i.e., $\hat{y}_{m,j}^\tau = y_j^\tau$, $\tau < t$) up to time slot t. Fig. 2 illustrates the context partition and the counter update. In time slot t, the accuracy $\hat{q}_m(\lambda)$ of DNN m for tasks with context in hypercube λ is estimated by:

$$\hat{q}_m(\lambda) = E_m^t(\lambda)/C_m^t(\lambda) \tag{6}$$

Consider a tasks x_j^t with context $\omega_j^t \in \lambda_j^t$, the estimated accuracy of DNN m for x_j^t is $\hat{q}_m(\lambda_j^t)$.

In each time slot, ADE is either in an exploration or exploitation phase. In the exploration phase, ADE selects a set of random DNNs that has not been selected often and aims to learn more precise accuracy for selected DNNs. In the exploitation phase, ADE chooses a set of DNNs which is expected to deliver the highest accuracy for admitted tasks. To determine the correct phase in a time slot, ADE first checks whether the context hypercubes have been sufficiently explored to give a precise accuracy estimation. For DNN m, its under-explored hypercubes are identified based on $C_m(\lambda)$:

$$\Lambda_m^{\text{ue},t} = \left\{\lambda \in \Lambda_T \;\middle|\; C_m^t(\lambda) < K(t)\right\} \tag{7}$$

where $K(t)$ is a control function to determine whether the context hypercube has been sufficiently explored. The function $K(t)$ needs to be appropriately designed to balance exploration and exploitation. Based on the under-explored hypercubes of DNNs $\Lambda_m^{\text{ue},t}, \forall m$ and the admitted tasks X^t in time slot t, we define under-explored DNNs in time slot t as:

$$\mathcal{M}^{\text{ue},t} = \left\{m \in \mathcal{M} \middle| \exists x_j^t \in X^t, \omega_j^t \in \lambda_j^t, \text{and } \lambda_j^t \in \Lambda_m^{\text{ue},t}\right\} \tag{8}$$

Algorithm 1 Adaptive DNN Ensemble with CC-MAB

Input: time horizon T, algorithm parameters h_T, $K(t)$;
Initialization create partition Λ_T; set $C_m(\lambda) = 0, E_m(\lambda) = 0, \forall \lambda \in \Lambda_T, m \in \mathcal{M}$;
for $t = 1, \ldots, T$ **do**
  Observe context $\omega_j^t \in \Omega$ of each task $x_j^t \in X^t$ and find context hypercube λ_j^t such that $\omega_j^t \in \lambda_j^t$ holds;
  Get $\mathcal{M}^{\text{ue},t}$ as defined in (8) and set $l^t = |\mathcal{M}^{\text{ue},t}|$;
  if $\mathcal{M}^{\text{ue},t} \neq \emptyset$ **then** ▷ *Exploration*
    if $l^t \geq B$ **then**
      $\mathcal{S}^t \leftarrow$ randomly select B DNNs in $\mathcal{M}^{\text{ue},t}$;
    else:
      $\mathcal{S}^t \leftarrow$ select all l^t DNNs in $\mathcal{M}^{\text{ue},t}$ and $B - l^t$ additional DNNs as in (9);
    end if
  else: $\mathcal{S}^t \leftarrow$ select B DNNs as in (10); ▷ *Exploitation*
  end if
  Observe the ground truth of the inference tasks;
  for each $m \in \mathcal{S}^t$ and $\lambda \in \{\lambda_j^t\}_{j=1}^{N^t}$ **do** ▷ *Updates*
    Update counters: $C_m^t(\lambda)$, $E_m^t(\lambda)$;
    Update estimation: $\hat{q}_m(\lambda) = E_m^t(\lambda)/C_m^t(\lambda)$;
  end for
end for

Given under-explored DNNs $\mathcal{M}^{\text{ue},t}$, ADE determines the phase of the current time slot.

Exploration: If the set of under-explored DNNs is non-empty, i.e., $\mathcal{M}^{\text{ue},t} \neq \emptyset$, ADE enters exploration. Let $l^t = |\mathcal{M}^{\text{ue},t}|$ be the number of under-explored DNNs. If $l^t \geq B$ DNNs, then ADE randomly selects B DNNs from $\mathcal{M}^{\text{ue},t}$. If $l^t < B$, ADE selects all l^t DNNs in $\mathcal{M}^{\text{ue},t}$. As budget B is not fully utilized, we pick $(B - l^t)$ additional DNNs that have the highest estimated accuracy for X^t:

$$s_b^t \in \underset{m \in \mathcal{M} \setminus \left\{\mathcal{M}^{\text{ue},t} \cup \left\{\cup_{i=1}^{b-1} s_i^t\right\}\right\}}{\arg\max} \frac{1}{N^t} \sum\nolimits_{j=1}^{N^t} \hat{q}_m(\lambda_j^t) \tag{9}$$

where $b = 1, \ldots, (B - l^t)$. If the DNN defined by (9) is not unique, ties are broken arbitrarily.

Exploitation phase: If the set of under-explored DNNs is empty, $\mathcal{M}^{\text{ue},t} = \emptyset$, ADE enters exploitation and selects B DNNs that have the highest estimated accuracy for X^t:

$$s_b^t \in \underset{m \in \mathcal{M} \setminus \left\{\cup_{i=1}^{b-1} s_i^t\right\}}{\arg\max} \frac{1}{N^t} \sum\nolimits_{j=1}^{N^t} \hat{q}_m(\lambda_j^t) \tag{10}$$

where $b = 1, \ldots, B$. At the end of the time slot, ADE observes the ground truth of tasks and updates the estimated accuracy for each selected DNN and hypercube.

C. Performance Analysis

The performance of ADE is measured by the reward loss compared to oracle, termed as *regret*. Let $\{\mathcal{S}^t\}_{t=1}^T$ be the DNN ensemble decision sequence of ADE, the regret is defined as:

$$R(T) = \mathbb{E}\left[\sum\nolimits_{t=1}^{T} U(X^t, \mathcal{S}^{*,t}, \boldsymbol{\mu}^t) - U(X^t, \mathcal{S}^t, \hat{\boldsymbol{q}}^t)\right]. \tag{11}$$

where $\boldsymbol{\mu}^t = \{\mu_m(\omega_j^t)\}_{\forall m,j}$ is the expected accuracy and $\hat{\boldsymbol{q}}^t = \{\hat{q}_m(\omega_j^t)\}_{\forall m,j}$ is the estimated accuracy learned by ADE at time slot t. A common designing goal of multi-armed bandit algorithms is to achieve a sublinear regret $R(T) = O(T^\gamma)$ with $\gamma < 1$. A sublinear regret bound guarantees the asymptotic optimality since $\lim_{T\to\infty} R(T)/T \to 0$. Next, we give a regret upper bound of ADE. The regret upper bound is derived based on the assumption that a DNN will have similar accuracy for inference tasks with similar context, which is formalized by the Hölder condition:

Assumption 1 (Hölder Condition)**.** There exists $L > 0$, $\alpha > 0$ such that $\forall m \in \mathcal{M}$ and $\forall \omega_j, \omega_k \in \Omega$, it holds that

$$|\mu_m(\omega_j) - \mu_m(\omega_k)| \leq L\|\omega_j - \omega_k\|^\alpha, \tag{12}$$

where $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^D$.

We note that this assumption is needed for the analysis of algorithm regret, but ADE can still be applied if it does not hold true. In that case, however, a regret bound might not be guaranteed. Under Assumption 1, the following theorem shows the regret order of ADE:

Theorem 1 (Bound for Regret $R(T)$)**.** Let $K(t) = t^{\frac{2\alpha}{3\alpha+D}}\log(t)$ and $h_T = \lceil T^{\frac{1}{3\alpha+D}} \rceil$. If ADE is run with these parameters and Assumption 1 holds true, the leading order of the regret $R(T)$ is $O\left(T^{\frac{2\alpha+D}{3\alpha+D}}\log(T)\right)$.

Proof. See Appendix A in the supplemental file. □

Theorem 1 indicates that the regret of ADE algorithm is sublinear in the time horizon T. Moreover, the bound is valid for any finite time horizon, providing a bound on the performance loss for any finite time slot. This can be used to characterize the convergence speed of ADE.

Remark on ground-truth retrieval: For some DL services, e.g., recommendation systems [28], the ground-truth of tasks can be directly retrieved. However, for other DL services, the retrieval of task ground-truths can be difficult. For example, to receive the ground-truth of image classification tasks, the service provider may need to use crowdsourcing services [29] such as Mechanical Turk and reCAPTCHA [1] to acquire labels from non-expert annotators. In this case, the ground-truth retrieval and update of accuracy estimation are delayed. Fortunately, some previous works [30] have investigated the impact of the delayed feedback on the performance of bandit algorithms, and show that the feedback with a bounded delay does not change the order of regret.

IV. Experiments

A. Experiment Setup

1) Edge Computing Platforms: A DELL workstation is used as an edge server, which is equipped with a 64-bit twelve Intel i7-8700K cores running at 3.70GHz, and two NVIDIA GeForce GTX 1080Ti GPUs. The experiment is run on Ubuntu 16.04 system with Tensorflow v1.5.0, cuDNN v7.0, CUDA v9.0. In each time slot, tasks are sent to the edge server in a batch-wise manner. The average size of task batches is 10.

[1]Mechanical Turk: www.mturk.com; reCAPTCHA: recaptcha.net.

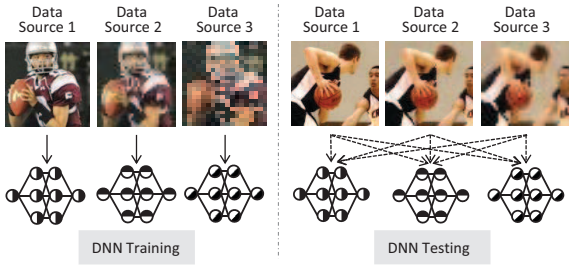

Fig. 3: Illustration of experimental setup.

		Test on HR	Test on LR-1	Test on LR-2
Train on HR	MobileNet	**0.631**	0.395	0.532
	Inception	**0.689**	0.426	0.606
	Incept-ResNet	**0.682**	0.464	0.595
Train on LR-1	MobileNet	0.264	**0.54**	0.427
	Inception	0.135	**0.541**	0.488
	Incept-ResNet	0.066	**0.592**	0.532
Train on LR-2	MobileNet	0.440	0.516	**0.60**
	Inception	0.511	0.551	**0.647**
	Incept-ResNet	0.551	0.583	**0.656**

TABLE I: Accuracy of DNNs

2) DL Service: Sport-Category Classification: We apply ADE to the sport-category classification problem. The DL service is to recognize common sport categories from individual scene images. The images are from the WIDER-attribute dataset [18]. We use 15 sport categories in the dataset and The application service on the edge server aims to classify sport categories for the received sport-related images. The WIDER-attribute dataset has three parts: training, validation, and test data. The training data contains 2,492 images that are used for training DNNs, the validation and test data contains 4,401 images that are used to evaluate our algorithm.

3) Training Available DNNs: The DNNs are different from each other in two aspects: 1) The data sources used for training DNNs and 2) The network structure used by DNNs. We pre-process the WIDER-attribute dataset in different ways to create different data sources. Three widely-used DNN structure: MobileNet-v2 [31], Inception-v3 [32], and Inception-ResNet-v2 [33], are used for training DNNs on the pre-processed WIDER-attribute dataset. We modify the resolution of images in the WIDER-attribute dataset to generate different training data. This setting is very realistic due to various types of image capturing devices, e.g., smartphones, surveillance cameras, SLR cameras. The image resolution is the context used by ADE. Each DNN structure has fixed input size for images, e.g., $224 \times 224 \times 3$ for MobileNet-v2, $299 \times 299 \times 3$ for Inception-v3, and $299 \times 299 \times 3$ for Inception-ResNet-v2. The resolution of original WIDER-attribute dataset is higher than the required dimension and we call it HR (high resolution) images. We create LR-1 (low resolution scale-1) data source by downsampling the original image to a size not exceeding

150×80, and LR-2 (low resolution scale-2) data source by downsampling the original image to a size not exceeding 300×150. We use HR, LR-1, and LR-2 to train MobileNet-v2, Inception-v2 and Inception-ResNet-v2 separately. In total, 9 models are trained and stored on the edge server. We evaluate the accuracy of each DNN on test data of HR, LR-1, and LR-2, the results are shown in Table I. There is a noticeable performance gap of DNNs across test datasets. We can see that DNNs have higher accuracy when the test data is from the same data source of the training data. This indicates that the task contexts can affect the performance of DNNs.

B. Performance Evaluation

We compare ADE with 4 benchmarks:

1) Oracle: Oracle knows the context-parameterized accuracy of DNNs and selects B DNNs that have the highest accuracy for the admitted tasks in each time slot.

2) Non-ensemble: This algorithm is a variant of ADE. It selects only one DNN in each time slot and is used as a baseline to validate the efficacy of DNN ensemble.

3) UCB1: UCB1 is a classic MAB algorithm that selected only one action in each time slot. To run UCB1 for DNN ensemble selection, we create super-arms, i.e., B-element combination of M DNNs. There will be a total of $\binom{M}{B}$ super-arms and UCB1 learns the reward of each super-arm.

4) Random: It randomly selects B DNNs in each time slot.

1) Accuracy Comparison: Fig.4 compares the time-averaging accuracy achieved by ADE and other 4 benchmarks. As expected, Oracle has the highest accuracy. Among the other algorithms, we see that ADE achieves higher accuracy for user tasks and its performance gradually converges to Oracle's performance. In addition, ADE outperforms UCB1 since it takes into account the context of inference tasks in DNN ensemble selection. In particular, it can be observed that ADE (69% accuracy) provides a $(69-62)/62 \times 100\% = 11.3\%$ accuracy improvement compared to Non-ensemble (62% accuracy). We can even see that Random has higher accuracy than Non-ensemble, indicating that using the DNN ensemble technique effectively helps improve the DL inference quality.

2) Impact of Ensemble Size: Fig.5 shows the regret achieved by ADE with different ensemble size B. We see that the regret decreases with the increase in ensemble size, which indicates that learning is not very necessary when the ensemble size is large. This is because the operator can include almost all DNNs in DNN ensembles with large ensemble size, and in this case, identifying best-fit DNNs becomes less useful. Moreover, we see that our algorithm is able to achieve sublinear regret with all ensemble sizes in the experiment. Fig. 6 shows the accuracy achieved by ADE after 4,400 time slots. The results show that higher accuracy can be achieved with large ensemble size but this accuracy improvement diminishes with the increase in ensemble size.

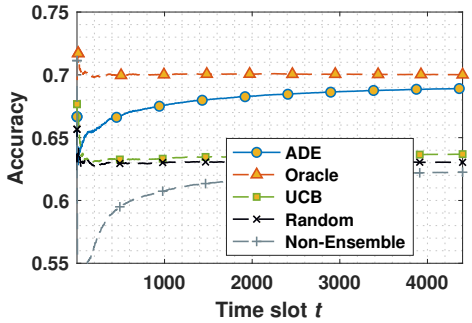

Fig. 4: Accuracy evolution ($B = 3$).

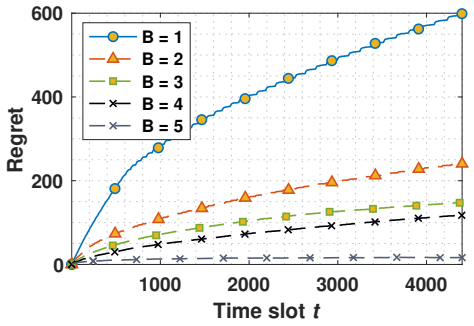

Fig. 5: Impact of B on regret.

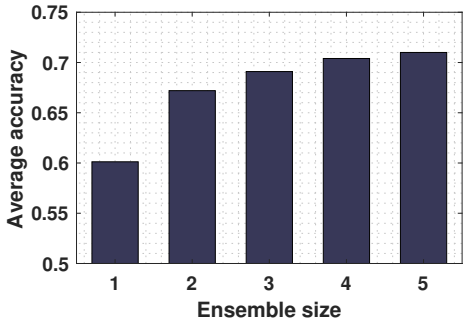

Fig. 6: Impact of B on accuracy.

C. Space and Time Complexity

We now show the space and time complexity of ADE during implementation. The implementation complexity is relevant to implementation schemes. Therefore, we will first show several determinants of implementation complexity, and then discuss efficient implementation schemes for ADE.

DNN Switching Cost and Implementation Schemes: A straightforward scheme to execute a DNN ensemble on edge server is to load and run DNNs one by one (referred to as LD-OBO scheme). This scheme frequently tears and reloads the DNNs in memory, which tends to incur a large delay for running DNN ensembles. We call this kind of delay as *intra-slot DNN switching cost*. Based on our experiment, loading a DNN into RAM incurs non-negligible delay, e.g., loading MobileNet requires 1.21 sec, loading Inception requires 2.05 sec, and loading Inception-ResNet requires 5.231 sec. This is for loading DNNs only, the time for running inference tasks has not been included. The intra-slot switching cost can be addressed by parallel execution of DNNs (referred to as LD-PAR) by loading all DNNs in the ensemble simultaneously

TABLE II: Memory Usage & Time Complexity of ADE with LD-NW and LD-W

Ensemble Size		$B=1$	$B=2$	$B=3$	$B=4$	$B=5$	$B=6$	$B=7$
Implementation Scheme (LD-NW)	**Memory Usage** (GB)	0.117	0.229	0.349	0.466	0.583	0.678	0.816
	Time Complexity (sec)	0.873 $\pm$0.650	1.014 $\pm$0.823	2.009 $\pm$1.001	3.102 $\pm$ 1.071	4.905 $\pm$ 1.219	5.786 $\pm$ 1.384	6.324 $\pm$ 0.707
Implementation Scheme (LD-W)	**Memory Usage** (GB)	1.083	1.080	1.081	1.077	1.107	1.092	1.099
	Time Complexity (sec)	0.024 $\pm$0.044	0.051 $\pm$0.119	0.098 $\pm$0.278	0.144 $\pm$0.396	0.178 $\pm$0.481	0.172 $\pm$0.459	0.175 $\pm$0.383

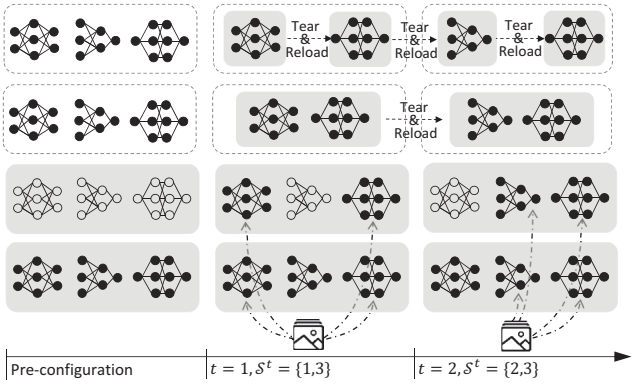

Fig. 7: Illustration of implementation schemes. *From top to bottom, the schemes are LD-OBO, LD-PAR, LD-NW, and LD-W. The gray background denotes the RAM. The network with hollow circles denotes the DNN structure, the solid circle indicates parameters. During pre-configuration, LD-OBO and LD-PAR does not use RAM, LD-NW only loads DNN structure, and LD-W loads both structures and parameters. LD-NW only load parameters for selected DNNs in each time slot.*

in the RAM at the beginning of each time slot. LD-PAR addresses the intra-slot switching cost. However, ADE changes its decisions across time slots. In this case, LD-PAR needs to reload a new DNN ensemble when ADE changes the DNN ensemble decision. We call this kind of delay as *inter-slot switching cost*. Due to the inter-slot switching cost, LD-PAR actually does not help much in reducing the inference delay if the DNN ensemble varies frequently across time slots.

A straightforward solution to avoid both intra-slot and inter-slot switching costs is to load all available DNNs before running ADE and run selected DNNs only in each time slot (referred to as LD-W). However, LD-W requires a large amount of memory since all available DNNs need to be loaded at the same time. To make our ADE suitable for edge servers with limited resources, we give a modified version of LD-W, referred to as LD-NW. LD-NW loads the network structures of all DNNs before running ADE, and in each time slot, the parameters of selected DNN are loaded for running task inferences. This weight loading process takes around 10 ms to 500ms depending on the DNN structure. This amount of delay is acceptable for most application services. Fig.7 illustrates all four implementation schemes, LD-OBO, LD-PAR, LD-NW, and LD-W, for comparison.

Performance comparison: Due to the space limitation, we only show the memory usage and inference delay for LD-NW and LD-W because the inference delay incurred by LD-OBO and LD-PAR is too large (over 10 secs with ensemble size $B = 2$). Table II shows the memory usage and time complexities of ADE when running with LD-NW and LD-W. We see that the memory usage of LD-NW is much less than that of LD-W. For example, when running ADE with ensemble size $B = 2$, the average memory usage of LD-W is 1.08 GB (with a total of 9 available DNNs), and the average memory usage of LD-NW is only 0.23 GB. However, LD-W incurs much lower inference time than that of LD-NW. In particular, running ADE with LD-W can achieve real-time responses, but it also requires large RAM resources. If the edge server is able to meet this resource requirement, then LD-W can directly used. If the computing resource of edge servers cannot support LD-W, the DL service provider should use LD-NW and pick an appropriate ensemble size for the trade-off between inference delay and resource usage.

V. Conclusions

In this paper, we applied DNN ensemble techniques to enhance DL inference performances on edge computing platforms. A unique problem called DNN ensemble selection is studied. The goal of DNN ensemble selection is to identify a subset of DNNs that works well for users' inference tasks and fits the limited computing capacity of edge servers. An online learning algorithm is proposed to solve the DNN ensemble selection problem. It learns the DNN accuracy based on the context of inference tasks and meanwhile selecting the best-fit DNN ensemble for admitted tasks. The proposed method is practical, effective, and easy to implement.

Appendix A
Proof for Theorem 1

Proof. Due to space limitation, we only provide a sketch of the proof. The regret can be divided into two parts: $R(T) = R_{\text{explore}}(T) + R_{\text{exploit}}(T)$, where $R_{\text{explore}}(T)$ and $R_{\text{exploit}}(T)]$ are the regrets incurred by exploration and exploitation, respectively. The regret of these two parts will be bounded separately. The key idea for bounding the exploration regret is to ensure that the number of time slots that ADE enters exploration phase is sublinear. Therefore, given that maximum utility loss in each exploration phase, the total regret incurred by exploration is sublinear. For the exploitation regret, we need to prove the gap between the utilities incurred by ADE and Oracle is sublinear in each time slot.

In fact, the leading order of $R(T)$ is determined by $R_{\text{explore}}(T)$ and therefore we only give detailed proof for bounding $R_{\text{explore}}(T)$. Let t be an exploration phase and then there exist at least a hypercube λ_j^t, $j = 1, \ldots, U^t$ and a DNN $m \in \mathcal{M}$ such that $C_m^t(\lambda_j^t) \leq K(t) = t^z \log(t)$. By the definition of $\mathcal{M}^{\text{ue},t}$, there can be at most $\lceil T^z \log(T) \rceil$ exploration phases in which DNN $m \in \mathcal{M}$ is selected to run inference tasks with context λ_j^t up to time horizon T. Since there are $(h_T)^D$ hypercubes in the partition, there can be at most $(h_T)^D \lceil T^z \log(T) \rceil$ exploration phases for DNN m in T time slot due to under-exploration. Additionally, there are M DNNs whose accuracy need to be learned in the exploration phase. Therefore, there can be at most $M(h_T)^D \lceil T^z \log(T) \rceil$ exploration phases in T time slots. In each exploration phase, the maximum number of inference tasks is $N^{\max}$ and hence the maximum loss is $1 \cdot N_{\max}$. The maximum loss exploration phases is $M(h_T)^D \lceil T^z \log(T) \rceil N_{\max}$, and the expected regret of exploration phase is bounded by:

$$\begin{aligned}\mathbb{E}\left[R_e(T)\right] \leq & M(h_T)^D \lceil T^z \log(T) \rceil N_{\max} \\ = & M \lceil T^\gamma \rceil^D \lceil T^z \log(T) \rceil N_{\max}.\end{aligned}$$

Using $\lceil T^\gamma \rceil^D \leq (2T^\gamma)^D = 2^D T^{\gamma D}$, it holds $\mathbb{E}\left[R_e(T)\right] \leq M N_{\max} 2^D (\log(T) T^{z+\gamma D} + T^{\gamma D})$. Using the values given in Theorem 1 for h_T and $K(t)$, we have $\gamma = \frac{1}{3\alpha + D}$ and $z = \frac{2\alpha}{3\alpha + D}$. This completes the proof. □

REFERENCES

[1] D. Sima, "Apples mobile processors," 2018.

[2] T. Lite, "Android to launch tensorflow lite for mobile machine learning," 2017.

[3] Core ml: Integrate machine learning models into your app. [Online]. Available: https://developer.apple.com/documentation/coreml.

[4] X. Xie and K.-H. Kim, "Source compression with bounded dnn perception loss for iot edge computer vision," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[5] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 184–199.

[6] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[7] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 331–344.

[8] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2018, pp. 389–400.

[9] R. F. Sean Hollister. (2016) How pokemon go aects your phones battery life and data. [Online]. Available: https://www.cnet.com/how-to/pokemon-go-battery-test-data-usage/

[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *arXiv preprint arXiv:1701.01090*, 2017.

[11] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[12] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[13] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[14] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[15] (2017) Large scale visual recognition challenge (ilsvrc). [Online]. Available: http://www.image-net.org/challenges/LSVRC/

[16] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[17] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019. [Online]. Available: https://arxiv.org/abs/1906.00091

[18] Y. Li, C. Huang, C. C. Loy, and X. Tang, "Human attribute recognition by deep hierarchical contexts," in *European Conference on Computer Vision*, 2016.

[19] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[20] H. Zhao, M. Pan, X. Liu, X. Li, and Y. Fang, "Exploring fine-grained resource rental planning in cloud computing," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 304–317, 2015.

[21] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine learning*, vol. 51, no. 2, pp. 181–207, 2003.

[22] S. Rothe and D. Söffker, "Comparison of different information fusion methods using ensemble selection considering benchmark data," in *2016 19th International Conference on Information Fusion (FUSION)*. IEEE, 2016, pp. 73–78.

[23] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[24] R. E. Schapire, Y. Freund, P. Bartlett, W. S. Lee *et al.*, "Boosting the margin: A new explanation for the effectiveness of voting methods," *The annals of statistics*, vol. 26, no. 5, pp. 1651–1686, 1998.

[25] H. Bonab and F. Can, "Less is more: a comprehensive framework for the number of components of ensemble classifiers," *IEEE Transactions on neural networks and learning systems*, 2019.

[26] K. Woods, W. P. Kegelmeyer, and K. Bowyer, "Combination of multiple classifiers using local accuracy estimates," *IEEE transactions on pattern analysis and machine intelligence*, vol. 19, no. 4, pp. 405–410, 1997.

[27] L. Chen, J. Xu, and Z. Lu, "Contextual combinatorial multi-armed bandits with volatile arms and submodular reward," in *Advances in Neural Information Processing Systems*, 2018, pp. 3247–3256.

[28] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "Drn: A deep reinforcement learning framework for news recommendation," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 167–176.

[29] J. Howe, *Crowdsourcing: How the power of the crowd is driving the future of business*. Random House, 2008.

[30] L. Chen and J. Xu, "Task replication for vehicular cloud: Contextual combinatorial bandit with delayed feedback," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 748–756.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[33] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.