TrustServing: A Quality Inspection Sampling Approach for Remote DNN Services

Xueyu Hou and Tao Han
Department of Electrical and Computer Engineering
University of North Carolina at Charlotte, North Carolina, USA
e-mail: {xhou,Tao.Han}@uncc.edu

***Abstract*—Deep neural networks (DNNs) are being applied to various areas such as computer vision, autonomous vehicles, and healthcare, etc. However, DNNs are notorious for their high computational complexity and cannot be executed efficiently on resource constrained Internet of Things (IoT) devices. Various solutions have been proposed to handle the high computational complexity of DNNs. Offloading computing tasks of DNNs from IoT devices to cloud/edge servers is one of the most popular and promising solutions. While such remote DNN services provided by servers largely reduce computing tasks on IoT devices, it is challenging for IoT devices to inspect whether the quality of the service meets their service level objectives (SLO) or not. In this paper, we address this problem and propose a novel approach named QIS (quality inspection sampling) that can efficiently inspect the quality of the remote DNN services for IoT devices. To realize QIS, we design a new ID-generation method to generate data (IDs) that can identify the serving DNN models on edge servers. QIS inserts the IDs into the input data stream and implements sampling inspection on SLO violations. The experiment results show that the QIS approach can reliably inspect, with a nearly 100% success rate, the service qualtiy of remote DNN services when the SLA level is 99.9% or lower at the cost of only up to 0.5% overhead.**

***Index Terms*—Edge Computing, AIoT, MLaaS, Cloud Computing**

I. Introduction

In recent years, deep neural networks (DNNs) are popular in various areas such as computer vision [1]–[5], autonomous vehicle [6], [7], and medical care [8]–[10]. A large number of DNN models have been designed for different functions with diverse performance requirements. As an example, many DNN models have been developed in the area of computer vision for image classification [1], [11]–[15], object detection [16]–[19], and segmentation [10], [20]. Moreover, a function, e.g., image classification in computer vision, can be realized by a collection of DNN models with varying accuracy-computational-complexity (Acc-$\mathcal{O}$) tradeoff as shown in Fig. 1. For example, the overall accuracy of AlexNet [11] on ImageNet dataset [21] is only 70% of that of Inception-V3 [12]. However, the number of operations, i.e., computational complexity $\mathcal{O}$, of Inception-V3 is over five times that of AlexNet. In most cases, DNN models with higher computational complexity usually show higher accuracy. Meanwhile, DNN models with higher computational complexity cost more for execution. Specifically, executing DNN models with higher computational complexity can affect two aspects: (1) higher execution latency: taking a longer time to execute with the same computing capability and the same amount of computing resources, and (2) higher execution cost: spending more computing resources, e.g., like computing cores of GPU and power consumption, to execute.

978-1-7281-6630-8/20/$31.00 2020 © IEEE

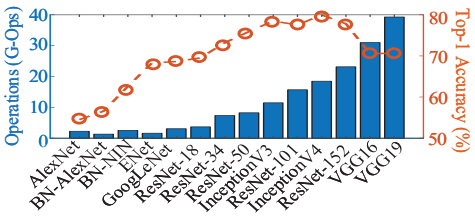


Fig. 1: *Comparison of Accuracy and Number of Operations over Classification DNN Models [22].*

Due to the high computational complexity of DNN models, it is challenging for IoT devices to execute DNN models on chip [23]–[25]. Under such a circumstance, remote DNN services provide IoT devices with rich computing resources on cloud/edge servers [23]–[26]. A basic remote DNN service structure is shown in Fig. 2. An IoT device can send a number of data (e.g., images) to a server for processing with DNNs (e.g., image classification). The server processes each datum with a DNN model (e.g., ResNet50 [1]) and sends the corresponding result (e.g., a class label) back to the IoT device. Such remote DNN services will be widely adopted in future AIoT era. For instance, IoT devices such as mobile phones, surveillance cameras, and wireless sensors may stream a huge amount of data (e.g., images, voices, etc.) to high-performance edge/cloud servers for processing with DNNs. The accuracy of DNN models will determine the Quality-of-Experience (QoE) of users and/or safety operations of IoT devices.

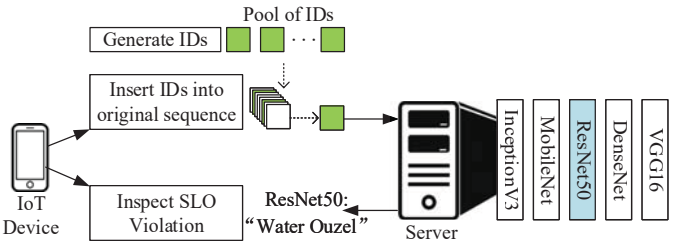
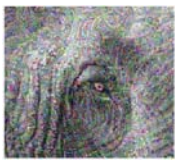


Fig. 2: *Overview of the QIS Approach in a Remote DNN Service System.*

Based on the development of DNN models and Machine-Learning-as-a-Service (MLaaS), we foresee that future remote DNN services will possess the following features: (1) multiple IoT devices share standardized DNN models to process their data, (2) servers provide a collection of standardized DNN models with different Acc-$\mathcal{O}$ tradeoffs (e.g., DNN models compared in Fig. 1), and (3) remote DNN services should maintain Service-Level-Agreement (SLA) between the service provider and its users. As the accuracy of DNN models determines the QoE and/or safety of IoT devices, we introduce the accuracy-based Service-Level-Objective (SLO) as a part of the SLA, which is the lower-boundary of DNN services' accuracy. If a DNN model's accuracy meets the accuracy-based SLO, using the DNN model to serve the user will not violate the SLA between the user and the service provider. From users' perspective, they expect servers to follow SLAs and provide remote DNN services meeting their SLOs to guarantee QoE and/or safety operations. However, to lower execution costs and/or to serve more IoT devices, servers may opportunistically switch to low-accuracy-low-computational-complexity DNN models, e.g., switching from Inception-V3 to AlexNet, which leads to the violation of the SLAs.

Detecting violations against accuracy-based SLO is challenging because the outputs of a DNN model usually contain no information about the model. Taking image classification as an example, an IoT device cannot figure out what DNN model was used to process an image purely based on the received class label, e.g., Water Ouzel as shown in Fig. 2. Alhough some machine learning verification approaches can verify a machine learning model based on cryptography, these approaches show high computational complexity and are only proven to work on relatively simple machine learning models such as support vector machine [27], shallow neural networks [27], [28], logistic regression [27], and k-Nearest Neighbor [27]. However, as multiple DNN models may be used during a remote DNN service session [23], [29], a low-cost approach to simultaneously identify multiple DNN models rather than to verify only one DNN model [30] is necessary to effectively inspect accuracy-based SLO violations. Thus, instead of focusing on the malicious attack against integrity of DNN models on servers [27], [28], [30], the focus of this paper is on inspecting the violation of the accuracy-based SLO caused by the DNN model switching among a set of DNN models with different Acc-$\mathcal{O}$ trade-offs in remote DNN services.

In this paper, we first develop an ID-generation method to effectively generate IDs for identifying among a set of DNN models. An **ID** is a specially-designed datum that can identify different DNN models with 100% accuracy simultaneously. Specifically, with an ID as input, different DNN models output different results. An example of a generated ID is shown in Fig. 3, the ID can identify {InceptionV3, MobileNet [13], ResNet50, DenseNet [31], VGG16 [32]} trained on ImageNet Dataset [21]. When receiving the corresponding result (output) of the ID from a server, IoT devices can find which DNN model was used to process the ID by looking up the table shown in Fig. 3. In practice, the IDs can be generated using the ID-generation method on a third-party platform that provides quality certification services for remote DNN services. Besides, the IDs can also be generated locally on IoT devices if they have sufficient computing resources. Note that the IDs can be generated offline before a remote DNN service session initializes, and our experiments show that the ID-generation method can generate a huge number of IDs efficiently.


Generated ID Image ($\tilde{x}$)

$\mathcal{D}_{candidate}$	Output
InceptionV3	Electric Ray
MobileNet	Brambling
ResNet50	Water Ouzel
DenseNet	Bullfrog
VGG16	Black Swan

Fig. 3: *An Example of an ID Generated with ID-Generation Method.*

Leveraging these IDs, we design a Quality Inspection Sampling (QIS) approach to inspect accuracy-based SLO violations as shown in Fig. 2. After a pool of IDs are generated by using the ID-generation method, an IoT device can acquire IDs from the pool and insert them into the original input data sequence. Based on the results of the IDs, the IoT device can identify which DNN model is currently used by the server in the remote DNN service. In this way, the QIS approach realizes inspection on accuracy-based SLO violations. For example, assume that an IoT device requires accuracy-based SLO to be higher than 0.78. With the QIS approach, the IoT device receives "Water Ouzel" as class label of the ID. Since "Water Ouzel" corresponds to ResNet50 and Acc(ResNet50)$<$ 0.78, the IoT device detects the accuracy-based SLO violation on the server. As servers are unaware of which data from IoT devices are IDs, they cannot treat IDs specially with high-accuracy DNN models. Our experiment shows that the QIS approach can reliably inspect the service qualtiy of remote DNN services when the SLA level is 99.9% or lower at the cost of only up to 0.5% overhead.

The contributions of this paper are:

- To the best of our knowledge, this is the first paper studying the service quality inspection method to ensure the accuracy-based SLO in remote DNN services for IoT.
- We design an ID-generation method that can effectively generate IDs to identify different DNN models running by remote servers.
- We develop the QIS approach for IoT devices to inspect potential accuracy-based SLO violations on servers in remote DNN services.
- We implemented the QIS approach and, through extensive experiments, validate the effectiveness of the QIS approach with almost zero overhead. We also evaluate the generalizability and efficiency of the ID-generation method.

This paper consists of the following sections: Section II briefly overviews features of remote DNN services. Section III describes the ID-generation method. Section IV integrates the ID-generation method into the QIS approach for service quality inspection. Section V experimentally verifies the generalizability of the ID-generation method and the effectiveness of the QIS approach. Section VI concludes the paper.

II. Features of Remote DNN Services

In this paper, we consider the remote DNN services that provide IoT devices with standardized DNN models [23]–[25] instead of requiring IoT devices to upload their customized DNN models. Specifically, we assume that the remote DNN services will have the following three features:

1) A set of standardized DNN models can be shared by a number of IoT devices: Before launching the remote DNN services, instead of training DNN models individually for each IoT device, the standardized DNN models are trained with the dataset from samples of all IoT devices [33]. For example, multiple autonomous vehicles can share DNN models trained by samples from the same city/area. Thus, instead of uploading customized DNN models to a server by each IoT device, servers use standardized DNN models to provide services to IoT devices. In each service session, an IoT device only needs send data to the remote server, and the server sends the processing results back to the IoT device using one of the standardized DNN models as shown in Fig. 2.

2) The DNN models exhibit diverse Acc-$\mathcal{O}$ tradeoffs: The diversity of the Acc-$\mathcal{O}$ tradeoffs can be caused by either different architecture of the DNN models or varying DNN model compression, e.g., pruning, in the model deployment [34]. For IoT devices, selecting a high-accuracy-high-computational-complexity DNN model leads to: (1) more reliable results as overall accuracy of such a DNN model is usually higher, e.g., Inception-V3, and (2) a greater expense in the DNN service because it consumes more computing resources on servers. For a server, serving an IoT device with a high-accuracy-high-computational-complexity DNN model results in: (1) a higher cost in execution, and (2) limited capability for serving a large number of IoT devices. Thus, both IoT devices and servers may adaptively select DNN models with different Acc-$\mathcal{O}$ tradeoffs. Specifically, IoT devices may select DNN models based on their accuracy, latency requirements, and budgets [29], [35]. Servers may select DNN models when serving an IoT device according to the number of active users and execution costs [23].

3) The remote DNN services are charged based on SLAs: A SLA consists of an accuracy-based SLO and consequences upon the violation against the SLO. IoT devices propose SLO when requesting remote DNN services to servers. In this paper, the accuracy-based SLO is defined as lower-boundary of accuracy of DNN services. On the one hand, a remote DNN service with a higher SLO, i.e., a higher lower-boundary of accuracy, cost more computing resources, and thus it charges more for using the service. On the other hand, if a violation against SLO happens with a high rate, e.g., $> 0.1\%$, servers are expected to pay service credits back to IoT devices. For example, according to Amazon S3 SLA [36], users will be paid by 10% service credits if the monthly uptime percentage is less than 99.9% but greater than or equal to 99.0%. The higher the violation rate, i.e., the lower the monthly uptime percentage, is, the more service credits will be paid to users.

Fig. 2 demonstrates the architecture of the remote DNN service systems. A set of candidate DNN models $\mathcal{D}_{candidate} = \{\bar{d}_1, \bar{d}_2, ..., \bar{d}_N\}$ are deployed on a server. These DNN models have different Acc-$\mathcal{O}$ trade-offs. Thus, servers can dynamically switch among the models in $\mathcal{D}_{candidate}$ during a remote DNN service session with an IoT device. Switching the DNN model affects accuracy of the DNN service. Once a remote DNN service is established, we denote the sequence of input data as $\hat{X}$, and each element of $\hat{X}$ is denoted as $\hat{x}_i$. The server uses a DNN model $\bar{d}_n \in \mathcal{D}_{candidate}$ to process an input datum $\hat{x}_i$, and sends corresponding result $\bar{d}_n(\hat{x}_i)$ back to the IoT device. In general, a remote DNN service with higher accuracy-based SLO requires more computing resources on servers.

Based on features of the remote DNN service systems, we design the QIS approach to inspect SLO violation as shown in Fig. 2. In the QIS approach, we first generate IDs that can 100% identify DNN models of $\mathcal{D}_{candidate}$ with the ID-generation method. Then, a selection of IDs ($\tilde{X}$) are inserted into the original sequence of input data ($\hat{X}$). We denote the hybrid input data sequence as X which contains both IDs and the original input data. The IDs in X implement sampling inspection against accuracy-based SLO violations on servers for IoT devices.

III. ID-Generation Method

The workflow of the ID-generation method is shown in Fig. 4. The target of ID-generation method is to generate IDs that deviate the outputs of a number of DNN models $\mathcal{D}_{candidate} = \{\bar{d}_1, \bar{d}_2, ..., \bar{d}_N\}$. Thus, N different references $\boldsymbol{O}_{ref} = \{\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_N\}$ can be preset and are utilized as the desired outputs for N different DNN models of $\mathcal{D}_{candidate}$. In other word, with an ID $\tilde{x}$ as an input datum, the output of DNN model $\bar{d}_n$ $(n = 1, 2, ..., N)$ is $\boldsymbol{o}_n$.

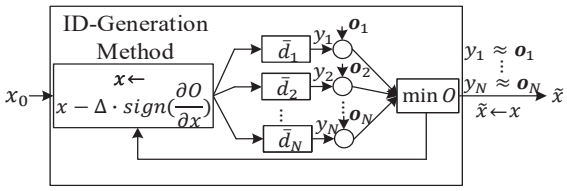


Fig. 4: *Workflow of ID-Generation Method.*

A. Problem Formulation

Definition 1. *Given a number of candidate DNN models $\mathcal{D}_{candidate} = \{\bar{d}_1, \bar{d}_2, ..., \bar{d}_N\}$ and randomly selected N different reference vectors $\boldsymbol{O}_{ref} = \{\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_N\}$, the generated ID $\tilde{x}$ is an input datum s.t.:*

$$\bigwedge_{n=1}^{N} \left(\bar{d}_n(\tilde{x}) = \boldsymbol{o}_n \right) \tag{1}$$

Definition 1 can be further relaxed to **Definition 2**:

Definition 2. *Given a number of candidate DNN models $\mathcal{D}_{candidate} = \{\bar{d}_1, \bar{d}_2, ..., \bar{d}_N\}$ and randomly selected N different reference vectors $\boldsymbol{O}_{ref} = \{\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_N\}$, $\exists \epsilon \in \Re^+$ and $\epsilon << 1$, such that the generated ID $\tilde{x}$ satisfies:*

$$\bigwedge_{n=1}^{N}\left(\left\|\bar{d}_n(\tilde{x})-\boldsymbol{o}_n\right\|<\epsilon\right) \tag{2}$$

where $\left\|\bar{d}_n(\tilde{x})-\boldsymbol{o}_n\right\|$ refers to $L1-Norm$ of vector $(\bar{d}_n(\tilde{x})-\boldsymbol{o}_n)$.

The length of the output vector of DNN model $\bar{d}_n$ is denoted as v_n $(n=1,2,...,N)$. Each element in reference output vector $\boldsymbol{o}_n$ is denoted as o_j^n, and each element in output vector $\bar{d}_n(x)$ is denoted as y_j^n $(j=1,2,...,v_n)$. Then, the problem of generating an ID can be formulated as:

$$\begin{aligned}&\min_{\tilde{x}}\frac{\sum_{n=1}^{N}\left(\sum_{j=1}^{v_n}\left\|o_j^n-y_j^n\right\|\right)}{N}\\&s.t. o_j^n\in\boldsymbol{o}_n,\\&\quad y_j^n\in\bar{d}_n(x).\end{aligned} \tag{3}$$

where $\left\|o_j^n-y_j^n\right\|$ refers to $L1$-Norm of $(o_j^n-y_j^n)$, which is the same as that in **Definition 2**. In other word, Eq. 3 adopts mean absolute error (MAE) to compute the objective function O. In Section V, MAE based objective function will be compared with other two types of loss functions, i.e., mean square error (MSE) and cross entropy (CE). Since y_j^n is a function of $\tilde{x}$ according to Eq. 3, the objective function O of Eq. 3 is also a function of x and the solution of Eq. 3 is to find optimal $\tilde{x}$ that minimizes the objective function O. In Section V, we will show that Eq. 3 is solvable experimentally, and for an optimal $\tilde{x}$, ϵ of Eq. 2 can be as small as 10^{-3}.

B. ID-Generation Method

Gradient descent is a popular way to train DNN models and to find solutions to optimization problems. Our ID-generation method adopts gradient descent to find a solution to the problem formulated in Eq. 3. Specifically, as shown in Fig. 4, the ID-generation method takes a randomly initialized input datum or a real-world input datum (e.g., a photo taken from real-world) as initial input x_0. At each iteration k, the ID-generation method computes the gradient of the objective function O of Eq. 3 to the input datum of current iteration k, i.e. $\frac{\partial O}{\partial x}|_k$. The datum x of current iteration k (denoted as $x^{(k)}$) is then modified based on $\frac{\partial \mathcal{O}}{\partial x}|_k$:

$$x^{(k+1)}=x^{(k)}-\Delta\cdot sign(\frac{\partial O}{\partial x}|_k) \tag{4}$$

where Δ is called $learning\ rate$ and is a hyper-parameter of gradient descent, and $sign(\cdot)$ denotes the sign function. We adopt $sign(\frac{\partial O}{\partial x}|_k)$ rather than $\frac{\partial O}{\partial x}|_k$ to accelerate the speed of gradient descent. The ID-generation method iterates the above procedure to modify an input datum x_0 according to Eq. 4 until every output $\bar{d}_i(\tilde{x})$ approximately equals to $\boldsymbol{o_i}$. In this way, our ID-generation method generates a modified input datum $\tilde{x}$, i.e., an ID, which satisfies **Definition 2**.

Details of the ID-generation method are described in Algorithm 1. The first input of Algorithm 1 is an initial seed input datum x_0, which is used as the initial input datum for the first iteration to calculate the first gradient. There is no constraint on the contents of x_0. It can be any input datum from real-world, or manually constructed input datum as our experiment shows in Section V. The second input of Algorithm 1 is a number of candidate DNN models $\mathcal{D}_{candidate}=\{\bar{d}_1,\bar{d}_2,...,\bar{d}_N\}$, which require a generated ID to identify one another. The third input of Algorithm 1 is N randomly selected reference output vectors $\boldsymbol{O}=\{\boldsymbol{o}_1,\boldsymbol{o}_2,...,\boldsymbol{o}_N\}$. The dimension of reference output vector $dim(\boldsymbol{o}_i)$ should be equal to the dimension of output vector of corresponding DNN model $dim(\bar{d}_n)$, i.e. $dim(\boldsymbol{o}_n)=dim(\bar{d}_n(x)), n=1,2,...,N$. There is no constraint on element values of reference output vectors either. The forth input of Algorithm 1 is the $learning\ rate$ Δ as in Eq. 4. It is a hyper-parameter that controls the converging speed of the ID-generation method, i.e., the number of iterations to generate an ID. In general, Δ of a very small value slows down the converging speed of gradient descent, and Δ of a very large value may lead the algorithm to miss optimal points. The selection of Δ is critical in controlling the convergence speed and will be further discussed in Section V. The fifth input of Algorithm 1 is another hyper-parameter ϵ, which is defined in **Definition 2**. The value of ϵ decides the difference between $\bar{d}_i(\tilde{x})$ and $\boldsymbol{o}_i$.

In Algorithm 1, line 3 to line 7 implements gradient descent to generate an ID $\tilde{x}$. When $\bar{d}_i(x^{(K)})\approx\boldsymbol{o}_i$ at iteration K (line 3), the loop stops iteration and outputs $x^{(K)}$ as $\tilde{x}$ (line 8). The function $CompObj(x,\boldsymbol{o},\bar{d})$ computes the objective function $O^{(k)}$ of each iteration k (line 10, 11). The function $CompObj(x,\boldsymbol{o},\bar{d})$ in Algorithm 1 adopts the same objective function of Eq. 3. The function $Speedup(\mathcal{G})$ takes the gradient $\mathcal{G}^{(k)}$ at iteration k as input and extracts the sign of $\mathcal{G}^{(k)}$ (line 12 to line 14), i.e., if an element of $\mathcal{G}^{(k)}$ is negative (positive), then the corresponding output element of $\mathcal{G}_m^{(k)}$ is -1 $(+1)$.

To learn which DNN model is running on servers, IoT devices can send an ID $\tilde{x}$ which is generated by the ID-generation method. By searching the returned result $\bar{d}_{server}(\tilde{x})$ in $\boldsymbol{o}_1,\boldsymbol{o}_2,...,\boldsymbol{o}_N$, the DNN model $\bar{d}_{server}$ running on the server can be identified. For example, an ID $\tilde{x}$ shown in Fig. 3 is generated to identify five DNN models, i.e., InceptionV3, MobileNet, ResNet50, DenseNet, and VGG16 corresponding to the class labels of "Electric Ray", "Brambling", "Water Ouzel", "Bullfrog", and "Black Swan", respectively. Thus, if the analysis result of $\bar{d}_{server}(\tilde{x})$ is "Brambling", an IoT device can identify that the serving DNN model on the remote server is MobileNet.

IV. Accuracy-Based SLO and QIS approach

When an IoT device employs a DNN service from a remote server, it expects to acquire a good-quality service. In general, there could be two types of Service Level Indicators (SLI) in remote DNN services: accuracy and latency. As we mainly focus on accuracy-based SLO in this paper, we will first define *nominal accuracy* which will be used as our SLI to define accuracy-based SLO. Then we will define accuracy-based SLO. Finally, we will describe our QIS approach which utilizes IDs generated by the ID-generation method to inspect accuracy-based SLO violations.

Algorithm 1: ID-Generation Method

Input: $x_0 \leftarrow$ randomly initial seed input matrix;
$\mathcal{D}_{candidate} \leftarrow$ candidate DNN models;
$\boldsymbol{O} \leftarrow$ randomly selected reference vectors;
$\Delta \leftarrow$ learning rate;
$\epsilon \leftarrow$ threshold value of $\left\|\bar{d}_n(x) - \boldsymbol{o}_n\right\|$

Output: $\tilde{x} \rightarrow$ ID.

——————-**Main Function**——————-

Initialize: $x \leftarrow x_0$;

while $\bigvee_{n=1}^{N}(\left\|\bar{d}_n(x) - \boldsymbol{o}_n\right\| \geq \epsilon)$ **do**

  $O \leftarrow CompObj(x, \boldsymbol{o}, \bar{d})$;

  $\mathcal{G} \leftarrow \partial O / \partial x$;

  $\mathcal{G}_m \leftarrow Speedup(\mathcal{G})$;

  $x \leftarrow x - \Delta \cdot \mathcal{G}_m$;

$\tilde{x} \leftarrow x$.

——————**Utility Functions**——————

Function $CompObj\,(x, \boldsymbol{o}, \bar{d})$ **:**

  $O \leftarrow \sum_{n=1}^{N}\left(\sum_{j=1}^{v_n}\left\|o_j^n - y_j^n\right\|\right)/N$;

  return O.

Function $Speedup\,(\mathcal{G})$ **:**

  $\mathcal{G}_m \leftarrow sign(\mathcal{G})$;

  return $\mathcal{G}_m$.

A. Nominal Accuracy

In most cases, we reference the accuracy of a DNN model $\bar{d}$ by the accuracy measured on validation dataset X_{eval} given corresponding ground-truth outputs to represent accuracy levels of DNN models, i.e.

$$Acc(\bar{d}) = \frac{\sum_{x_i \in X_{eval}} Cmp\big(\bar{d}(x_i), \bar{y}(x_i)\big)}{|X_{eval}|} \tag{5}$$

where $Acc(\bar{d})$ represents accuracy of $\bar{d}$, $|X_{eval}|$ represents the number of data in validation dataset, and $\bar{y}(x_i)$ represents the ground-truth of a datum x_i in X_{eval}. $Cmp(\cdot,\cdot)$ compares the output of $\bar{d}$ and ground-truth of x_i, e.g. $Cmp(\bar{d}(x_i), \bar{y}(x_i))$ can be equal to 1 if $\bar{d}(x_i)$ is equal to $\bar{y}(x_i)$ and to 0 if $\bar{d}(x_i)$ is not equal to $\bar{y}(x_i)$ in classification applications. The accuracy values in Fig. 1 are measured on validation dataset of ImageNet [21]. A straightforward observation is that a DNN model with higher computational complexity $\mathcal{O}$ usually shows higher accuracy. Thus, for any application, a set of N candidate DNN models $\mathcal{D}_{candidate} = \{\bar{d}_1, \bar{d}_2, ..., \bar{d}_N\}$ with accuracy-computational-complexity (Acc-$\mathcal{O}$) trade-off exists. The DNN models in $\mathcal{D}_{candidate}$ can be either different architectures like DNN models in Fig. 1 or one architecture with different compression (e.g., pruning) levels.

In a remote DNN service scenario, after establishing a DNN service relationship with a server, an IoT device starts to send data to the server for DNN processing like shown in Fig. 2. We denote the time slot when streaming starts as t_1, and the time slot when streaming ends as t_L. In most cases, these data can be described as an L-$length$ sequence within the streaming period $[t_1, t_L]$, where $t_l \in [t_1 : t_L]$ represents each time slot of sending a datum x_l from the IoT device to the server:

$$X[1:L] = \{x_1, x_2, ..., x_L\} \tag{6}$$

However, for a datum $x_l \in X[1:L]$ and the corresponding DNN model $d^{(t_l)}$ processing it on the server, the accuracy cannot be strictly calculated with Eq. 5 because no ground-truth output of x_l is given. Thus, we define *nominal accuracy* for $\forall x_l \in X[1:L]$ with $Acc(d^{(t_l)})$ calculated based on Eq. 5:

$$Acc^{(nom)}(x_l) = Acc(d^{(t_l)}) \tag{7}$$

In remote DNN services, the DNN models in $\mathcal{D}[1:L]$ are selected from $\mathcal{D}_{candidate}$, i.e.

$$d^{(t_l)} \in \mathcal{D}_{candidate}, \forall d^{(t_l)} \in \mathcal{D}[1:L] \tag{8}$$

where $\mathcal{D}[1:L]$ represents the sequence of DNN models used to process each datum of $X[1:L]$. For convenience, we will call *nominal accuracy* as *accuracy* and use Acc to represent $Acc^{(nom)}$ in this paper, and Acc will be our SLI for defining accuracy-based SLO.

B. Accuracy-Based SLO

In general, IoT devices can set a lower-boundary $\bar{Acc}$ of accuracy for processing each datum in $X[1:L]$:

$$d^{(t_l)} \geq \bar{Acc}, \forall t_l \in [t_1 : t_L] \tag{9}$$

We call the above inequality as *accuracy-based SLO*. Such accuracy-based SLO still allows servers to dynamically assign different DNN models as long as the accuracy values of utilized DNN models are above $\bar{Acc}$. For example, according to Fig. 1, if an IoT device sets an accuracy-based SLO with $\bar{Acc}$ of 0.7, then servers can utilize ResNet34, ResNet50, and InceptionV3 to serve the IoT device without violating accuracy-based SLO while servers cannot utilize ResNet18 or AlexNet to serve the IoT device as the accuracy of these DNN models are below 0.7. In reality, IoT devices may even vary the value of $\bar{Acc}$ based on different QoE and/or safety requirements during a DNN service session. Once an IoT device changes $\bar{Acc}$, there will be different subsets of DNN models in $\mathcal{D}_{candidate}$ that are forbidden from serving the IoT device on servers. Further, SLA can be made based on accuracy-based SLO. For example, if a server utilizes DNN models that violate accuracy-based SLO at a rate of over 0.1% or higher during a remote DNN service session with an IoT device, it will reimburse some percentage of the fee back to the IoT device.

C. QIS approach

The QIS approach is composed of three steps:

Step 1: Generating IDs Given $\mathcal{D}_{candidate}$, IDs can be generated with the ID-generation method. Besides identifying each DNN model of $\mathcal{D}_{candidate}$, IDs should satisfy: (1) IDs are in the same size as other input data $\hat{x} \in \hat{X}$, such that IDs cannot be figured out by servers. (2) IDs can be generated as many as possible. The first requirement prevents servers from processing IDs specially with high-accuracy DNN models.

The second requirement prevents servers from receiving an ID repeatedly and identifying the ID by comparison with the historical record of received data.

Step 2: Inserting IDs into original input data sequence $\hat{X}$ After initializing a remote DNN service, an IoT device can insert K different IDs $\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_K\}$ into original input data sequence $\hat{X}$. We denote the hybrid data sequence as $X[1:L]$, which contains both $\hat{X}$ and $\tilde{X}$. IoT devices know specifically the index positions of IDs in $X[1:L]$ but servers are unaware of which of them are IDs. We denote the indexes of IDs in $X[1:L]$ as a set of integers $\{z_1, z_2, ..., z_K\}$, where $z_1 < z_2 < ...z_K$. Thus, an IoT device can learn what type of DNN model is used for processing each datum of $\{x_{z_1}, x_{z_2}, ..., x_{z_K}\}$, and the DNN model sequence is denoted as $\{d_{z_1}, d_{z_2}, ..., d_{z_K}\}$.

Step 3: Inspecting accuracy-based SLO violations Inspecting accuracy-based SLO violations through the QIS approach is straightforward, i.e., when an IoT device finds that:

$$\exists k \in [1, K], s.t. Acc(d_{z_k}) < \bar{Acc} \tag{10}$$

the IoT device can confirm that the server violates accuracy-based SLO because Eq. 9 is no longer satisfied. In Eq. 10, $Acc(d_{z_k})$ is calculated by Eq. 5. In Section V, we will see that our QIS approach can effectively inspect accuracy-based SLO violations that happen at a rate of 0.1% or above.

V. Experiment

In this section, the performance of the ID-generation method and the QIS approach is evaluated via extensive experiments.

A. Computational Cost of ID-Generation Method

To study the computational cost of the ID-Generation Method, we set up our experiment on a computer with NVIDIA GeForce RTX 2070 and the CUDA version is 10.0. We use Pytorch 1.2.0 deep learning framework. For experiments, we use pre-trained DNN models for classification as $\mathcal{D}_{candidate}$. These models are pre-trained on ImageNet dataset [21] which contains 1000 classes. We randomly select different class (label) as output reference, i.e., $\boldsymbol{o}_i$ in Algorithm 1, for each DNN model of $\mathcal{D}_{candidate}$. We utilize a softmax-layer at the end of each candidate DNN model to normalize outputs to a 1000-$length$ confidence vector, and each element of the confidence vector is between 0 and 1. In this way, we can convert an output reference of a label to a 1000-$length$ confidence vector. For example, if we select an output reference of a label as "tiger shark", then the corresponding 1000-$length$ confidence vector is a vector with the fourth element as 1 and all the other elements as 0. With these 1000-$length$ confidence vectors as reference outputs, we can utilize the ID-generation method in Algorithm 1 with Eq. 2 and Eq. 3, in which we set $\epsilon = 0.001$.

In Fig. 5 (a), we show how *learning rate* (i.e., Δ in Algorithm 1) affects the number of iterations of Algorithm 1 in generating an ID. Fig. 5 (b) shows the effect of loss function formula (i.e., $CompObj(x, \boldsymbol{o}, \bar{d})$ in Algorithm 1) on the number of iterations. Fig. 5 (c) and (d) further study the generalizability of the ID-generation method across a different number of candidate DNN models and various initial seed input images.

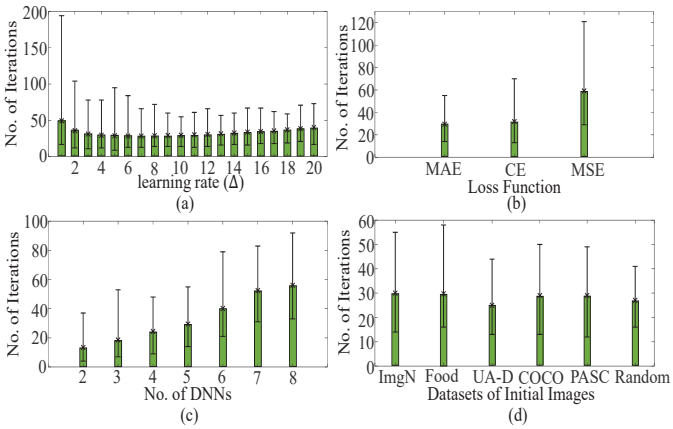


Fig. 5: *Computational Cost of ID-Generation Method: (a) Number of Iterations vs. Learning Rate, (b) Number of Iterations vs. Loss Function, (c) Number of Iterations vs. Number of Candidate DNNs, (d) Number of Iterations vs. Datasets of Initial Images.*

1) Learning Rate: We first use five classification DNN models for the study of *learning rate*. The five DNN models are Inception-V3, ResNet50, ResNet34, ResNet18, and AlexNet. As shown in Fig. 1, we have $Acc(Inception-V3) > Acc(ResNet50) > Acc(ResNet34) > Acc(ResNet18) > Acc(AlexNet)$, and $\mathcal{O}(Inception-V3) > \mathcal{O}(ResNet50) > \mathcal{O}(ResNet34) > \mathcal{O}(ResNet18) > \mathcal{O}(AlexNet)$. Thus, these five DNN models are with *Acc-$\mathcal{O}$* trade-off. We randomly select five different labels as reference outputs of an ID, i.e., $o_{Inception-V3}$ ="Electric Ray", $o_{ResNet50}$ ="Brambling", $o_{ResNet34}$ ="Water Ouzel", $o_{ResNet18}$ ="Bullfrog", $o_{AlexNet}$ ="Black Swan", which means that an ID is expected to be classified to a different label by different DNN models. We vary *learning rate* Δ in Algorithm 1 from 1 to 20 as shown in Fig. 5 (a). For images, the *learning rate* Δ represents the pixel value change (can be either $+\Delta$ or $-\Delta$, which depends on the sign of gradient as shown in line 13 of Algorithm 1) per iteration, e.g., $\Delta = 10$ represents that the pixel value changes by 10 per iteration. The test takes ImageNet Validation Dataset [21] as initial input images x_0, which contains 50,000 images. We record the maximum number, average number, and the minimum number of iterations for each learning rate value over the 50,000 images. We can see that all the 50,000 initial images can be modified by the ID-generation method to an ID that can be classified to a different label by different DNN models with *learning rate* from 1 to 20. As shown in Fig. 5 (a), a small pixel value change per iteration (i.e., Δ) leads to a large number of iterations to find an optimal output (generating an ID). For example, with *learning rate* of 1, the number of iterations to generate an ID can be as high as 190, even almost 50 on average. On the other hand, a large pixel value change per iteration tends to increase the number of iterations to find an optimal output. For example, the number of iterations keeps increasing with the increase of *learning rate*

from 10 to 20. We can also find that small pixel value change leads to a large range of number of iterations. For example, with *learning rate* of 1, the maximum number of iterations is 190 and the minimum number of iterations is only 10. With *learning rate* of 10, it shows the smallest range of number of iterations, i.e., the maximum number of iterations is 55 and the minimum number of iterations is 14. Moreover, the average number of iterations is 30 with *learning rate* of 10. The *learning rate* from 6 to 10 shows a very similar average number of iterations. We finally select *learning rate* of 10 in the following experiments because it shows the most stable and relatively small number of iterations with the ImageNet validation dataset as initial images.

2) Loss Function: In Algorithm 1, we use Mean-Absolute-Error (MAE), i.e., L-1 Norm in line 10 of Algorithm 1. In Fig. 5(b), we compare the loss function of MAE with two other types of loss function formula, i.e., Mean-Square-Error (MSE) and Cross-Entropy (CE). Again, the test is over 50,000 images of ImageNet Validation Dataset as initial images x_0, and the candidate DNN models are the same as the test of *Learning Rate*. We can see that all the 50,000 initial images can be modified by the ID-generation method to an ID that can be classified to a different label by different DNN models with all the three types of loss functions. We find that MAE shows the best performance compared with the other two. And MSE shows the highest number of iterations to generate an ID. Thus, it is reasonable to adopt MAE (L-1 Norm) in the ID-generation method.

3) Number of Candidate DNNs: To further verify the generalizability of our ID-generation method, we study across different numbers of candidate DNNs with settings of *learning rate* as 10 and loss function as MAE. Specifically, we use the set of {Inception-V3, ResNet18} as 2-candidate-DNN case, the set of {Inception-V3, ResNet34, ResNet18} as 3-candidate-DNN case, the set of {Inception-V3, ResNet50, ResNet34, ResNet18} as 4-candidate-DNN case, the set of {Inception-V3, ResNet50, ResNet34, ResNet18, AlexNet} as 5-candidate-DNN case, the set of {Inception-V3, ResNet50, ResNet34, ResNet18, AlexNet, MobileNet-V2} as 6-candidate-DNN case, the set of {Inception-V3, ResNet50, ResNet34, ResNet18, AlexNet, MobileNet-V2, SqueezeNet} as 7-candidate DNNs case, the set of {Inception-V3, ResNet50, ResNet34, ResNet18, AlexNet, MobileNet-V2, SqueezeNet, ShuffleNet} as 8-candidate DNNs case. And we randomly select different labels as reference outputs of an ID, i.e., $o_{Inception-V3}$ ="Electric Ray", $o_{ResNet50}$ ="Brambling", $o_{ResNet34}$ ="Water Ouzel", $o_{ResNet18}$ ="Bullfrog", $o_{AlexNet}$ ="Black Swan", $o_{MobileNet-V2}$ ="Tebetan terrier", $o_{SqueezeNet}$ ="Academic Gown", $o_{ShuffleNet}$ ="One-Armed Bandit", which means that an ID is expected to be classified to a different label by different DNN models. In Fig. 5(c), we can see that all the 50,000 initial images can be modified by the ID-generation method to an ID that can be classified to a different label by different DNN models for all the seven sets of candidate DNN models. With the increase of the number of candidate DNN models to identify, the number of iterations to generate an ID also increases. For example, it takes only 12 iterations on average to generate an ID for the 2-candidate-DNNs case, and it takes 55 iterations on average to generate an ID for the 8-candidate-DNN case.

4) Datasets of Initial Input Images: We also verify the generalizability of our ID-generation method regarding initial images. We test on five datasets, i.e., ImageNet [21], Food101 [37], UA-DETRAC [38], COCO [39], and PASCAL [40]. Moreover, we generated 1,000 images with each pixel value randomly generated between 0 and 255, which is denoted as "Random" in Fig. 5(d). The *learning rate* is 10 and the loss function is MAE. And the candidate DNN models are the same as the tests of *Learning Rate* and *Loss Function*. In Fig. 5(d), we can see that the average number of iterations with all the six datasets of initial images is between 25 and 30, and the maximum number of iterations is between 40 and 60. Thus, we infer from the experimental results that the ID-generation method does not constrain the space of initial images. For images as input, the ID-generation method can modify any natural images taken from real-world to IDs, or any artificial images generated. For example, given the input image size of $(224 \times 224 \times 3)$, ID-generation method can generate IDs from all $256^{150,528}$ images. As reference outputs can be randomly selected, for each initial image, more than one ID can be generated given different reference outputs. On the other hand, the experimental results also show the efficiency in generating an ID compared with searching for an ID directly among $256^{150,528}$ images in brute force.

With our testbed, we find that, for the case of 5 candidate DNN models, the latency per iteration is around 26ms. Thus, for 5 candidate DNN models, it takes less than 0.8s on average to generate an ID (as the average number of iteration is 30). To conclude, we can see that our ID-generation method is highly robust against hyper-parameter settings (e.g., learning rate and loss function). Moreover, the ID generation method can generate IDs with a wide range of initial images. With appropriate hyper-parameter settings, we can generate IDs with any initial images efficiently. The experiment shows that our ID-generation method can generate IDs as many as possible efficiently, and the generated ID can identify different DNN models in $\mathcal{D}_{candidate}$ with 100% accuracy as long as the reference output (e.g., labels in classification) of each DNN model in $\mathcal{D}_{candidate}$ is selected differently from one another. One note is that, though our experiment shows results on image classification models, the ID-generation method can also be utilized to other types of DNNs.

B. Success Rate of QIS Approach

As an ID can identify different DNN models in $\mathcal{D}_{candidate}$ with 100% accuracy, the success rate of inspecting accuracy-based SLO violations is purely determined by *Step 2* and *Step 3* of the QIS approach. Specifically, IoT devices can decide how many IDs to insert and where to insert them into the original input data sequence $\hat{X}$. In our experiment, we use the checking rate (CR) to represent the average ratio of the number

of IDs over the total number of input data. For example, $CR = 0.01$ represents that there is one ID inserted per 100 input data, and $CR = 0.005$ represents that there is one ID inserted per 200 input data. For a fixed length of an input data sequence, the lower the CR is, the fewer the number of IDs is inserted. We assume that IoT devices can insert an ID at any position of input data sequence independently with the same probability of CR. Thus, the insertion of IDs follows Poisson Distribution $P(CR)$, where CR is the average number of IDs per unit time (per input datum). With $P(CR)$, we can generate traces of hybrid data sequence $X[1:L]$. Specifically, we select five different CR values, i.e., 0.01, 0.005, 0.001, 0.0005, and 0.0001. For each CR value, we generate 1000 traces of hybrid data sequence $X[1:100000]$ of 100,000 input data (including IDs). In other words, given a CR value, the number of IDs in $X[1:100000]$ can be estimated by $CR \cdot 100000$. For example, given $CR = 0.0001$, there are roughly 10 IDs in $X[1:100000]$.

To evaluate the success rate of the generated hybrid data sequence, we assume that the probability of the server switching to DNN models that do not satisfy Eq. 9 also follows Poisson Distribution $P(\lambda)$. Specifically, we assume that λ vary from 0.0001 to 0.001 by a step of 0.0001 as shown in x-axis of Fig. 6. $\lambda = 0.0001$ means that accuracy-based SLO violation happens once per 10,000 input data on average and $\lambda = 0.001$ means that accuracy-based SLO violation happens once per 1,000 input data on average. Moreover, we test with different length $l = 1, 10, 20, 30$ (number of sequential input data) of accuracy-based SLO violations. For example, if $l = 10$ and accuracy-based SLO violation happens on processing the k-th input datum, then the input data from the k-th to the $(k+9)$-th will be processed by a DNN model that does not satisfy Eq. 9. As any ID can identify the type of DNN model with 100% accuracy, an IoT device can successfully inspect accuracy-based SLO violations of Eq. 10 if at least one ID is sent during accuracy-based SLO violation happening on the server, i.e., at least one ID is between the k-th input datum and the $(k+l-1)$-th given k is one of the positions where accuracy-based SLO violation starts to happen. As long as a one-time slot of accuracy-based SLO violation event is inspected by an ID, an IoT device can confirm that the server violates accuracy-based SLO, which can be regarded as a successful inspection. For each l and λ, we also generate 1,000 traces of DNN model sequence $\mathcal{D}[1:100000]$. Thus, the success rate is the ratio of the number of successful inspection events over 1,000 times of tests.

Fig. 6(a) shows the success rate of inspection to average violation happening rate (λ) in case of $l = 1$, which means that violation happens discretely per input datum processing. With the increase of checking rate (i.e., higher ID insertion frequency), the success rate also increases. Given a checking rate of 0.01 (i.e., one ID inserted per 100 input data), the QIS approach can inspect violation against SLA level of 99.95% with almost 100% success rate (the rate of accuracy-based SLO violation is over 0.05%). Given a checking rate of 0.005 (i.e., one ID inserted per 200 input data), the QIS approach

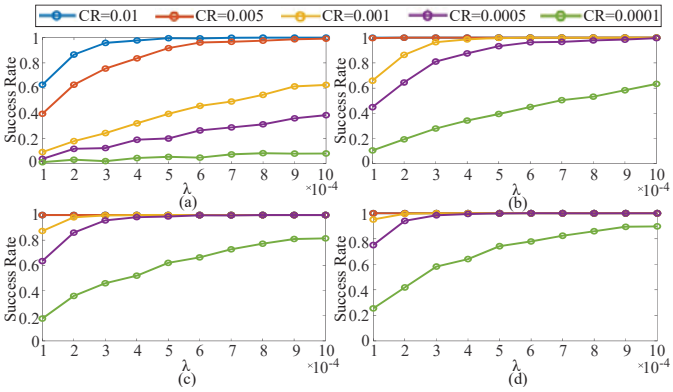


Fig. 6: *Success Rate of the QIS Approach vs. Rate of Accuracy-Based SLO Violation: (a) Discrete SLO Violation, (b) Length of SLO Violation is 10, (c) Length of SLO Violation is 20, (d) Length of SLO Violation is 20.*

can inspect violation against SLA level of 99.91% with almost 100% success rate (the rate of accuracy-based SLO violation is over 0.09%). In general, the QIS approach with a checking rate of 0.005 can successfully inspect violations against SLA level of 99.9%. Fig. 6(b) shows the success rate of inspection against average violation happening rate (λ) in case of $l = 10$, which means that once violation happens, it will keep on processing 10 input data in sequence. As more input data are processed by illegal DNN models that do not satisfy Eq. 10, the QIS approach can inspect violations with lower checking rate. In general, the QIS approach with a checking rate of 0.0005 (i.e., one ID inserted per 2,000 input data) can successfully inspect violations against SLA level of 99.9%. Further, in case of $l = 20$ (Fig. 6(c)), the QIS approach with a checking rate of 0.0005 (i.e., one ID inserted per 2,000 input data) can successfully inspect violations against SLA level of 99.95%. And in case of $l = 30$ (Fig. 6(d)), the QIS approach with a checking rate of 0.0005 (i.e., one ID inserted per 2,000 input data) can successfully inspect violations against SLA level of 99.96%.

Note that we test on a sequence of 100,000 input data. For example, if an IoT device sends images to a server with 30FPS, then it will send over 100,000 images to the server within one hour. Thus, our QIS approach can successfully inspect the service qualtiy of remote DNN services when the SLA level is 99.9% or lower within an hour.

VI. Conclusion

In this paper, we have presented a quality inspection sampling approach for remote DNN services and developed an ID-generation method. The ID-generation method can generate IDs to identify a set of DNN models with 100% accuracy. Experiments show that a huge number of IDs can be efficiently generated with low computational consumption locally or on a third-party platform that provides quality certification services for remote DNN services. Given a pool of IDs generated by the ID-generation method, the QIS approach randomly selects several IDs from the pool. By inserting these IDs randomly into the sending data sequence, the QIS approach can reliably

inspect the service qualtiy of remote DNN services when the SLA level is 99.9% or lower at the cost of only up to 0.5% overhead.

ACKNOWLEDGEMENT

This work is partially supported by the US National Science Foundation under Grant No. 1731675, No. 1810174, and No. 1910844.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and S. Jian, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 39, no. 4, pp. 664–676, 2014.

[3] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," *Advances in Neural Information Processing Systems*, vol. 1, 2015.

[4] Y. H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[5] C. C. Dan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22*, 2011.

[6] N. Engel, S. Hoermann, M. Horn, V. Belagiannis, and K. Dietmayer, "Deeplocalization: Landmark-based self-localization with deep neural networks," in *IEEE Intelligent Transportation Systems Conference - ITSC*, 2019.

[7] M. R. Heinen, F. S. Osorio, F. J. Heinen, and C. Kelber, "Seva3d: Using artificial neural networks to autonomous vehicle parking control," in *Neural Networks. IJCNN '06. International Joint Conference on*, 2006.

[8] S. K. Pandey and R. R. Janghel, "Recent deep learning techniques, challenges and its applications for medical healthcare system: A review," *Neural Processing Letters*.

[9] S. Spnig, A. Emberger-Klein, J.-P. Sowa, A. Canbay, and D. Heider, "The virtual doctor: An interactive clinical-decision-support system based on deep learning for non-invasive prediction of diabetes," *Artificial Intelligence in Medicine*, vol. 100, p. 101706, 2019.

[10] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P. M. Jodoin, and H. Larochelle, "Brain tumor segmentation with deep neural networks," *Medical Image Analysis*, vol. 35, pp. 18–31, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, no. 2, 2012.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *arXiv preprint arXiv:1512.00567*, 2015.

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *arXiv:1801.04381v4*, 2018.

[14] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *arXiv:1707.01083v2*, 2017.

[15] F. Iandola, S. Han, W. Moskewicz, K. Ashraf, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5MB model size," *arXiv:1602.07360v4*, 2016.

[16] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems*, pp. 379–387, 2016.

[17] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, Proceedings, Part I*, pp. 21–37, 2016.

[19] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Las Vegas, NV, USA, June 27-30*, pp. 779–788, 2016.

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[22] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *CoRR*, vol. abs/1605.07678, 2016.

[23] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: an approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys , Singapore*, pp. 123–136, 2016.

[24] C. Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, "Toward computation offloading in edge computing: A survey," *IEEE Access*, vol. 7, pp. 131543–131558, 2019.

[25] Z. Fang, D. Hong, and R. K. Gupta, "Serving deep neural networks at the cloud edge for vision applications on mobile platforms," in *Proceedings of the 10th ACM Multimedia Systems Conference, MMSys, Amherst, MA, USA*, pp. 36–47, 2019.

[26] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *IEEE Conference on Computer Communications (INFOCOM)*, April 2018.

[27] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, X. Lin, S. Hu, and M. Du, "Veriml: Enabling integrity assurances and fair payments for machine learning as a service," *CoRR*, vol. abs/1909.06961, 2019.

[28] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, Long Beach, CA, USA*, pp. 4672–4681, 2017.

[29] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," *CoRR*, vol. abs/1911.04946, 2019.

[30] Z. He, T. Zhang, and R. B. Lee, "Sensitive-sample fingerprinting of deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Long Beach, CA, USA, June 16-20*, pp. 4729–4737, 2019.

[31] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Honolulu, HI, USA, July 21-26*, pp. 2261–2269, 2017.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, Conference Track Proceedings*, 2015.

[33] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS, Fort Lauderdale, FL, USA*, pp. 1273–1282, 2017.

[34] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015.

[35] Q. Liu and T. Han, "DARE: Dynamic adaptive mobile augmented reality with edge computing," in *IEEE 26th International Conference on Network Protocols (ICNP)*, Sep. 2018.

[36] "Amazon aws s3 sla," *https://aws.amazon.com/s3/sla/*.

[37] "Food101 dataset," *http://www.vision.ee.ethz.ch/datasets$_e xtra/food-101/$*.

[38] L. Wen, D. Du, Z. Cai, Z. Lei, M. Chang, H. Qi, J. Lim, M. Yang, and S. Lyu, "UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking," *arXiv CoRR*, vol. abs/1511.04136, 2015.

[39] "Coco dataset," *http://cocodataset.org/*.

[40] "Pascal dataset," *http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2010/*.