NNFacet: Splitting Neural Network for Concurrent Smart Sensors

Jiale Chen, Duc Van Le, Rui Tan, and Daren Ho

Abstract—Various deep neural networks (DNNs) including convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have shown appealing performance in various classification tasks. However, due to their large sizes, a single DNN often cannot fit into the memory of resource-constrained smart IoT sensors. This paper presents a DNN splitting framework called *NNFacet* that aims to run a DNN-based classification task on a total of N concurrent battery-based sensors observing the same physical process. We begin with determining the importance of all CNN filters or RNN units in learning each class. Then, an optimization problem divides the class set into N subsets and assigns them to the sensors, where the important CNN filters or RNN units associated with a class subset form a small model that is deployed to a sensor. Lastly, a multilayer perceptron is trained and deployed to a cloud or edge server, which yields the final classification result based on the low-dimensional features extracted by the sensors using their small models for the same observation. We apply NNFacet to three case studies of voice sensing, vibration sensing, and visual sensing. Extensive evaluation shows that NNFacet outperforms four baseline approaches in terms of system lifetime, latency, and classification accuracy.

Index Terms—Internet of Things, distributed DNN inference, speech recognition, vibration analysis, video analytics

1 Introduction

Off-the-shelf deep neural network (DNN) models such as VGGNet [1], ResNet [2], and DeepSpeech [3] have been increasingly employed in Internet of Things (IoT) systems for achieving satisfactory performance in many applications such as the speech recognition, image classification and segmentation. However, due to their heavy weights, the execution of such advanced DNN models often incurs high computing overhead and memory usage. For instance, the VGG-19 consists of more than 21 million parameters and requires about 241 MB memory. While offloading the executions of advanced DNN models to a cloud or edge server is possible, it introduces several issues including high latency and communication bandwidth usage in transmitting the raw sensor data, privacy concerns, and poor scalability, especially when wireless communication is adopted.

Running the DNNs at the smart IoT sensors is desirable to enable the real-time and privacy-preserving data analytics [4]. Recent studies [5]–[11] have proposed various DNN compression techniques to reduce the memory usage and computation overhead. However, there exists a fundamental trade-off between the inference accuracy and model size. Compressing a large DNN model into one that can fit into the limited memory of a smart sensor or meet the computation overhead constraint may lead to poor inference accuracy. This paper considers a scenario of *concurrent* smart sensors that simultaneously observe the same physical process. This scenario provides a different dimension to address the memory and computation constraints by letting each sensor executes a portion of the DNN model. The considered concurrent sensors setting can be found in reality. For instance, in a smart home, the microphones embedded in appliances and smartphones form concurrent sensors. As another example, to transcribe a meeting, the attendants' smartphones form concurrent sensors. Similarly, in the industrial environment, multiple concurrent vibration sensors can be deployed to monitor the status of a machine.

To develop a solution perpendicular to model compression and for the setting of concurrent sensors, this paper proposes a DNN model splitting framework, called *NNFacet*, that decomposes a large multi-class DNN into multiple small models. Each of these small models is responsible for learning a subset of the classes and deployed on a resource-constrained concurrent smart sensor. The smart sensors collaboratively perform distributed inference by running their models and fuse the results to generate a final inference result. The distributed inference enables *in situ* advanced data analytics with low latency and sharing of inference workload to reduce the energy usage of each sensor.

To realize NNFacet, we formulate an optimization problem that divides the multi-class DNN into multiple size-constrained class-specific models, while maximizing the system lifetime subject to the requirement on classification accuracy achieved by the class-specific models. This problem consists of two coupled sub-problems with combinatorial complexities: 1) how to assign the classes to the concurrent sensors, and 2) how to assign the CNN filters or RNN units to the class-specific models. Specifically, as the number of filters or RNN units is often in the order of thousands and much larger than the numbers of classes and sensors, the filter/unit assignment sub-problem introduces dominating complexity. To solve this sub-problem, NNFacet begins with determining the importance of all filters or RNN units for learning each class. The filters or RNN units with importance indexes higher than a pre-defined threshold are assigned to form each class-specific model. Then, given the filter/unit assignment solution, the original

Jiale Chen and Duc Van Le are with HP-NTU Digital Manufacturing Corporate Lab, Nanyang Technological University, Singapore.
Rui Tan is with School of Computer Science and Engineering, Nanyang Technological University, Singapore.
Daren Ho is with HP Inc.

model splitting problem is reduced to the class assignment problem which can be solved using existing optimization tools within acceptable time. We also propose a greedy algorithm to efficiently find the class assignment solution, especially when the number of classes and sensors is large. Lastly, NNFacet employs a multilayer perceptron that runs on a cloud or edge server and fuses the results of the class-specific models to generate the final classification result. NNFacet is periodically executed to update the class-specific models running on the sensors to adapt to their battery residuals for maximizing system lifetime.

We conduct experiments using real edge devices based on six public datasets to evaluate the effectiveness of NNFacet for three sensing tasks, i.e., speech recognition, vibration analysis, and low-power video analytics. The first two are based on real concurrent sensors, while the last one applies virtual concurrent sensors running on the same real wireless camera to process consecutive image frames with similar contents in parallel. The evaluation results for these three case studies show that NNFacet always achieves longer system lifetimes and higher memory reductions, while maintaining the classification accuracy and latency, compared with four baseline approaches proposed in our prior work [12] and two existing works [9], [10]. Moreover, compared with the original multi-class DNN, NNFacet significantly reduces per-sensor memory requirement and execution latency.

Our preliminary work [12] designed a heuristic algorithm to split a multi-class CNN with the main objective of satisfying the memory constraint of the smart sensors. Based on [12], this paper makes the following contributions:

- We formulate the model splitting problem and identify challenges in decomposing the multi-class DNN into multiple class-specific models for concurrent smart sensors.
- We propose NNFacet that leverages the DNN filter pruning and optimization techniques to efficiently address the formulated problem. We design NNFacet for the two major forms of DNNs, i.e., CNNs and RNNs.
- We conduct extensive experiments with six public datasets in three sensing modalities to evaluate the effectiveness of NNFacet and show the performance gains compared with four baseline approaches.

The remainder of this paper is organized as follows. §2 reviews related work. §3 presents the background and problem formulation. §4 describes the design of NNFacet. §5 presents evaluation results. §6 concludes this paper.

2 Related Work

In this section, we review the related works on the DNN compression and splitting.

■ **DNN compression:** A number of studies [5]–[11] have proposed various DNN compression approaches that allow running DNN models on embedded devices with limited computing resources. The studies in [5]–[8] focused on general methods of pruning the model parameters (e.g., weights) to reduce the memory and computation resources required to store and run the DNN models. For instance, Denton *et al.* [5] applied singular value decomposition to compress convolutional layers in a CNN, and then fine-tuned the approximated layers to restore the accuracy. Han *et al.* [6] compressed a CNN model by removing the redundant connections with small weights. Then, the remaining weights are quantized to enforce weight sharing among multiple connections, and reduce the number of bits representing each connection. They can reduce the memory sizes of AlexNet and VGGNet by 53x and 49x, respectively. Shi *et al.* [7] developed a fine-grained compressed structured block pruning technique that can compress various RNNs with a higher pruning rate up to $25\times$ without accuracy loss, compared with the baseline RNN compression techniques. Chatzikonstantinou *et al.* [8] proposed an RNN pruning approach that considers the weight matrices as a collection of time-evolving signals and iteratively prunes the weights with similar temporal dynamics to reduce the model size. These parameter pruning approaches can reduce the required memory size of CNNs and RNNs by pruning the number of model parameters. However, these approaches do not reduce the number of filters in the convolutional layers, which, however, are the most computation-intensive part of a CNN.

The studies in [9]–[11] focused pruning the filters that play the role of the feature extractors in the convolutional layers. Pruning these filters significantly reduces the size and computation overhead. For instance, Fang *et al.* [11] proposed a framework called NestDNN that includes a triplet response residual method to rank the importance of filters across the convolutional layers. Then, the NestDNN iteratively prunes the unimportant filters and retrains the pruned model to compensate pruning-induced accuracy loss. The iteration stops when the pruned CNN cannot provide the minimum accuracy required by the designer. Yao *et al.* [9] proposed the DeepIoT framework that keeps the minimum number of non-redundant filters while maintaining the same accuracy as that of the original CNN. Alippi *et al.* [10] pruned a CNN model by keeping a few front convolutional layers only, and replaced the fully connected and softmax layers with a trained classifier (e.g., decision tree). Moreover, they further reduced the CNN's computation overhead by removing unimportant filters.

The above studies focused on compressing the large DNN model into one model with reduced memory size and computation overhead. However, the compression in general results in inference accuracy loss when the compression ratio exceeds a certain limit. Thus, applying them to meet the smart sensors' memory constraints may result in unacceptable accuracy reduction. Differently, our work focuses on splitting the multi-class DNN into multiple class-specific models that can fit into the smart sensors. The proposed NNFacet only prunes unimportant filters from the class-specific model whose memory and energy usages exceed the sensor's capacities. In §5, we compare the total model size and inference accuracy achieved by NNFacet with those of the two approaches in [9], [10]. The evaluation results show that NNFacet achieves a better trade-off between model size and accuracy for resource-constrained smart sensors.

■ **DNN splitting:** The study [13] has shown the feasibility of decomposing a multi-class DNN model into multiple binary models, each of which is sensitive to the samples in a single class. Different from [13], we consider the task of decomposing a multi-class model into an appropriate number of the class-specific models to meet the smart sensors' computing resource constraints and maximize the system lifetime, based on detailed modeling of DNN memory and energy usages. Each class-specific model given by NNFacet is sensitive to the samples of multiple classes. We also employ late fusion to fuse the outputs of the class-specific models to yield the final inference result. Moreover, the studies in [14], [15] proposed various collaborative inference approaches, in which a complex DNN model is partitioned into multiple sequential parts to be deployed on the edge, fog, and cloud computing nodes. Each computing node executes a DNN partition that captures all data classes. Differently, in our work, each node executes a small model that captures a subset of classes.

Our preliminary work [12] has presented the design of a heuristic algorithm that splits a multi-class DNN with the main objective of reducing the memory sizes of the class-specific models. It does not consider the energy constraint of the smart sensors, which, however, is important for battery-based smart sensors. In this paper, we make the following new contributions. First, we analyze DNN memory/energy usages and formulate a problem that aims at maximizing the system lifetime, subject to accuracy requirement, memory and energy constraints of the smart sensors. The solution to the formulated problem is different from the solution in [12] that addresses memory constraint only. Second, we conduct new experiments to compare the prior approaches including [12] and NNFacet proposed in this paper. The evaluation results show that NNFacet achieves more memory and energy savings while maintaining inference accuracy and latency. Third, this paper addresses both CNNs and RNNs, while our prior work [12] only considers CNNs.

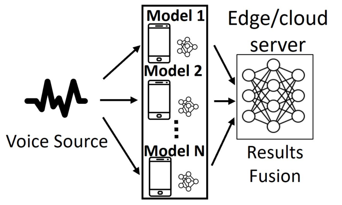
3 Background and Problem Formulation

In this section, we describe three applications that involve concurrent sensors as illustrated in Fig. 1. Then, we present the problem statement and assumptions of this study. Next, we present the DNN memory and energy models that are used in the problem formulation. Lastly, we formulate the DNN splitting problem and discuss the challenges in solving the problem.

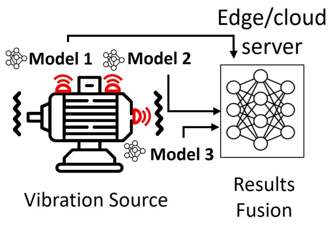
3.1 Applications with Concurrent Sensors

3.1.1 Voice recognition

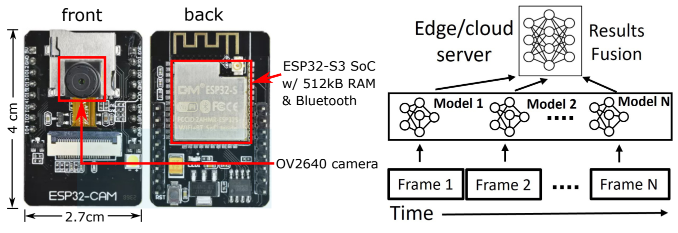
DNNs have been widely adopted for voice recognition applications including keyword spotting [16] and automatic speech recognition [3], [17]. The IoT applications based on human voice interactions often require real-time response for good user experience. Thus, the approach of transmitting the audio data to the cloud for speech recognition is not desirable due to the uncertain and possibly long communication latency. On the other hand, the speech recognition DNNs like DeepSpeech [3] are of large sizes and may not fit into a single IoT device. NNFacet is a promising approach to address this tussle for the scenarios with concurrent smart microphones. In an application of automatic transcription, the attendants of a meeting can use their smartphones as



(a) Concurrent microphones.



(b) Concurrent vibration sensors.



(c) Real-time video analytics based on RAM-constrained camera.

Fig. 1. Three applications with concurrent sensors. (a) Concurrent microphones in a smart home environment to perform keyword spotting or automatic speech recognition collaboratively. (b) Concurrent vibration sensors deployed to monitor the condition of a machine in factory. (c) The full deep model that cannot fit into the limited RAM of the wireless camera can be decomposed into multiple small class-specific models. These small models, which are loaded to the camera RAM sequentially to process the consecutive image frames with similar content, form *virtual* concurrent sensors.

the concurrent sensors to share the workload of automatic speech recognition, achieve good timeliness, and save battery energy. In smart homes, microphones are increasingly integrated into home appliances, e.g., television sets, set-top boxes, smart fridges, smart speakers, and etc. As the smart home appliance interoperability advances, the microphones on the home appliances may form concurrent sensors to capture the same user's voice and collaboratively perform keyword spotting or automatic speech recognition.

3.1.2 Vibration analysis

Vibration is an important sensing modality in various monitoring applications. The vibration signals of industrial machines (e.g., motors, assemblers, and conveyors) and structures reflect their internal states. Malfunction of the vibrating objects and structural changes usually result in variations in the vibration amplitude and frequency [18]. Thus, vibration analysis is important to machine/structure condition monitoring and predictive maintenance. Vibration signals have also been exploited for activity recognition [19], because structural vibration signals contain information of human activities. Meanwhile, DNNs have been used for vibration analysis. For instance, in [20], [21], VGG-19 and LetNet-5 are trained to detect and classify faults of rotating machines. LetNet-5 achieves 99.79% accuracy in detecting machine faults. In [22], RNNs achieve an error rate of 0.56% in detecting the faults of electrical transformers. These systems deploy multiple vibration sensors to increase the information about the monitored process.

The existing DNN-based detectors and classifiers are often executed on a centralized resourceful node. However, when wireless vibration sensors are adopted, collecting the high-rate vibration measurement data from the sensors to the centralized node is challenging. The industrial spaces typically have noisy and time-varying wireless conditions due to the moving parts of the production lines and electromagnetic noises from the working machines. Thus, it is desirable to perform the fault detection and classification locally at the sensors without transmitting the raw vibration data. NNFacet can be employed for the concurrent vibration sensors deployed on the same machine. The *in situ* distributed inference can avoid the energy-intensive raw data transmissions and achieve low-latency detection.

3.1.3 In situ video analytics on low-power smart camera

Today, wireless smart cameras have been widely deployed for visual sensing applications, including traffic control and industrial activity monitoring. Without replying on cables for power supply and network connectivity, the wireless cameras can be deployed in an *ad hoc* fashion when demanded. In addition, smart camera has been adopted for navigation of small-size autonomous drones [23] and insect-scale robots [24]. Due to the limited power supply, these smart cameras in general have limited computation resources. For instance, the ESP32-CAM as shown in Fig. 1(c), which is a wireless camera platform supporting the TensorFlow Lite Micro, has only 512 kB static random access memory (RAM). Thus, a complex DNN may not fit into the smart camera's limited RAM. Based on the observation that the consecutive image frames in a video trace captured by the camera have similar contents, one possible way to deal with the limited RAM problem is to decompose the complex DNN into multiple class-specific small models and load them sequentially into the RAM to process the consecutive image frames respectively. The results of the small models are fused to generate the video analytics result. In this method as illustrated in Fig. 1(c), the perception processes based on the small models form *virtual* concurrent sensors running on the same smart camera.

3.2 Approach Overview and Assumptions

We consider a large DNN model for a classification task with multiple learning classes. The main goal of the proposed NNFacet is to split this multi-class DNN model into multiple class-specific models which can fit into the memory-constrained concurrent sensors while maximizing the system lifetime subject to the classification accuracy requirement. Moreover, we aim to design NNFacet for the two major forms of DNNs which are CNNs and RNNs.

Specifically, a CNN model generally consists of four types of layers, i.e., convolutional, pooling, activation, and fully connected (FC) layers. Among these types, the convolutional layers are the most computation-intensive. A convolutional layer contains a set of three-dimensional (3D) filters, which extract invariant local 2D features. An FC layer contains a set of neurons connected to each input. Pruning the filters and neurons helps reduce both the overheads for storing the model parameters in RAM and executing the layer. Moreover, a filter has unique impact on learning each data class [13]. Thus, NNFacet aims to form the class-specific CNN models, each of which only contains the filters and neurons important to its responsible classes. Different from CNNs, RNNs do not use the 3D convolutional filters for feature extraction. An RNN is often formed by a chain of repeating RNN units, each of which has a chunk of multiple neural network layers. Each layer often contains multiple neurons which are the most compute-intensive elements in the RNNs. For example, a long short term memory (LSTM) network unit generally has three sigmoid layers and one tanh layer. Therefore, NNFacet keeps the important neurons of these neural network layers to form the class-specific RNN models.

TABLE 1
Effects of distance on voice classification accuracy.

Distance (meter)	0.02	0.5	1	1.5
Accuracy (%)	80	78	61	49

In this study, we assume that the input data of the class-specific models collected by the concurrent sensors have the same/similar contents. The real deployments may deviate from this assumption due to sensor characteristics and/or the dynamics of the monitored process. For instance, in the speech recognition systems, the recordings of the concurrent microphones may have different voice qualities caused by the heterogeneities in their hardware and surrounding environments. As a result, the class-specific DNN models on these microphones can yield reduced and varied performance. Transfer learning [25] and data augmentation [26] techniques have been applied to mitigate the impact of these issues.

Furthermore, the distance between the microphone and the voice source may also affect the accuracy achieved by the DNN models [27]. We conduct an experiment to evaluate such effect in terms of the accuracy of the VGG-19 [1] on the LibriSpeech [28] dataset. The details of the implemented VGG-19 model and dataset are described in §5.1. Specifically, we select 20 speech samples of two words "from" and "have" from the dataset, and use a Huawei P30 Pro mobile phone as a speaker to replay each sample ten times. Meanwhile, a Maono AU-A04TR microphone is used to record a total of 200 speech samples from the mobile phone at a certain distance. The recorded samples are fed into the VGG-19 model which is pre-trained using the original LibriSpeech dataset. Table 1 presents the voice recognition accuracy of the VGG-19 over 200 testing samples under various distances from 0.02 to 1.5 meters. From Table 1, we can see that the accuracy decreases with the distance. The distance-aware DNNs proposed in [27] can be used to mitigate the impact of the distance. Our proposed NNFacet can be applied to split such advanced DNNs to multiple class-specifical models. Another feasible solution is to deploy the microphones within a pre-determined distance (e.g., 0.5 meters in our experiment) from the voice source such that their recordings are strong enough and have similar contents to achieve desired accuracy.

The above solutions to address the microphone and environment inconsistency issues can be also adopted for the acoustic-based vibration analysis. Moreover, the concurrent vibration sensors are often attached on the monitored object. Thus, they can likely record similar vibration signals, especially when monitoring the solid objects. For the video analytics based on the virtual concurrent sensors, the class-specific models are used to process the consecutive image frames captured by the same camera. The video profiling method proposed in [29] can be used to evaluate the temporal correlation between the consecutive frames. Then, the camera's frame rate is set to an appropriate value such that the input frames of the class-specific models have the similar contents, i.e., the high temporal correlation.

3.3 DNN Memory and Energy Models

This section models the memory and energy usages of a CNN and discusses how these models can be varied to address RNN. We assume that the classi-specific model i to be deployed on sensor i consists of subsets of filters F_i and FC neurons R_i from the original model. It also includes all the pooling, activation of the original model.

Let m_i and e_i denote the amounts of memory and energy usage to run the model i and to complete the inference per sample, respectively. We adopt the modeling method in [30] that considers the m_i as the total number of bytes required to store all parameters of the convolutional and FC layers of the model i. Note that the pooling and activation layers do not have learnable parameters that need RAM space. Let s_f and I_f denote the kernel size and the number of input channels of a filter $f \in F_i$. Then, the number of parameters of the filter f is $I_f(s_f^2+1)$. The number of parameters of an FC neuron is equal to the number of inputs of this neuron, denoted by I_r. Given the model i's sets of filters F_i and neurons R_i, the m_i is calculated as

$$m_i = b\left(\sum_{f\in F_i} I_f(s_f^2+1) + \sum_{r\in R_{\rm i}} I_r\right), \tag{1}$$

where b denotes the number of bytes required to store a parameter.

We follow the CNN energy modeling method in [31] to model the per-sample energy usage e_i as the sum of the energy used for the inference computation and memory access, denoted by $e_{c,i}$ and $e_{m,i}$, respectively, i.e., $e_i = e_{c,i} + e_{m,i}$. Specifically, the $e_{c,i}$ is estimated based on the number of multiply-accumulate operations (MACs) for executing the filters and FC neurons of the model i to predict the label of an input sample. The number of MACs required to execute a filter with the kernel size of s_f is calculated as $s_f^2 I_f h_f w_f$, where h_f and w_f are the height and width of the filter's output feature map, respectively. The number of MACs for executing an FC neuron is also equal to the number of the neuron inputs I_r. Then, the $e_{c,i}$ is expressed as

$$e_{c,i} = e_{\rm mac}\left(s_f^2 \sum_{f\in F_i} I_f h_f w_f + \sum_{r\in R_{\rm i}} I_r\right), \tag{2}$$

where $e_{\rm mac}$ denotes the energy usage per MAC. Note that the value of $e_{\rm mac}$ depends on the processor. It is often estimated as $e_{\rm mac} = 2/n_{\rm flopsw}$ [32], where $n_{\rm flopsw}$ is the number of floating point operations per second per watt (FLOPS/W) of the processor. For instance, for the Raspberry Pi-4B single-board computer equipped with an ARM Cortex-A72 processor, the $n_{\rm flopsw}$ is 1.35 GFLOPS/W [33]. Thus, for the Raspberry Pi-4B, we set $e_{\rm mac} = 1.48\,\rm nJ$.

The memory access energy usage $e_{m,i}$ of the model i is the total energy used for loading the model i with a memory size of m_i to into RAM, and accessing the model i's activations of convolutional and FC layers from RAM. Specifically, the $e_{m,i}$ can be estimated as [34]:

$$e_{m,i} = e_{\rm mem}\left(m_i + b|F_i|h_f w_f + b|R_i|\right), \tag{3}$$

where $e_{\rm mem}$ is the amount of energy used for accessing a data bit in RAM; $e_{\rm mem} m_i$ is the total energy used for loading the model into RAM; $e_{\rm mem} b(|F_i|h_f w_f + |R_i|)$ is the total energy used for accessing the model from RAM. The $e_{\rm mem}$ is often modeled as $139 \times e_{\rm mac}$ [34].

By setting $F = \emptyset$ and letting R denote the set of RNN units, the above modeling in Eqs. (1)-(3) addresses RNN.

3.4 Problem Formulation

Let C and S denote the sets of the data classes and the concurrent sensors, respectively, where the number of sensors is not greater than the number of classes, i.e., $|S| \le |C|$. Let the F and R denote the sets of filters and FC neurons of the original CNN models. The main goal is to split the original CNN model into at most $|S|$ class-specific models. Thus, NNFacet's model splitting problem consists of two aspects. First, to identify $C_i \subset C$ and assign it to sensor i. Second, to identify $F_i \subset F$, $R_i \subset R$ and assign them to sensor i. The model i can classify the input sample into $|C_i| + 1$ classes, where the $|C_i|$ classes are from C_i and the remaining one class is a *null class* that represents all classes in $C \setminus C_i$. The main objective is to maximize the lifetime of the system consisting of the concurrent sensors S by assigning the classes and filters/neurons subject to the energy and memory constraints of every sensor as well as the inference accuracy requirement.

The model splitting problem, denoted by OPT-1, is solved periodically to adapt the system to the sensors' latest battery residual levels. Each sensor $i \in S$ has certain amounts of available memory and residual energy, denoted by M_i and E_i, respectively, at the beginning of each *assignment period*. We assume that a total of L inference samples are processed by the system in each assignment period. Then, the sensor i's residual energy after the next assignment period is calculated as $E_i - L\cdot(e_{c,i} + e_{m,i})$, where $e_{c,i}$ and $e_{m,i}$ are the per-sample energy usages for the inference computation execution and memory access, respectively. We define the system lifetime as the operational time of the system until the first sensor runs out of energy. Then, the objective function of OPT-1 is to maximize the minimum among the remaining energies of the sensors after the next assignment period, i.e., $\min_{i\in S}\{E_i - L\cdot(e_{c,i}+e_{m,i})\}$, where the $\min\{.\}$ represents the minimum operator.

Moreover, the OPT-1 has the following four constraints. First, the total energy required to run the class-specific model i, i.e., $L\cdot(e_{c,i}+e_{m,i})$, should be less than or equal to the sensor i's remaining energy level of E_i. Second, the total memory usage m_i should not be more than the sensor's available memory M_i. Third, the accuracy, denoted by a_L, for the fusion result of all class-specific models over L inference samples should be equal to or higher than the inference accuracy requirement, denoted by $A_{\rm req}$. Finally, each class $j \in C$ should not be assigned to more than one model.

We define a binary decision variable x_{ij} as follows. If the model i deployed to sensor i is sensitive to the class $j \in C$, $x_{ij} = 1$; otherwise, $x_{ij} = 0$. Similarly, a binary variable y_{ik} is defined to represent whether the model i contains

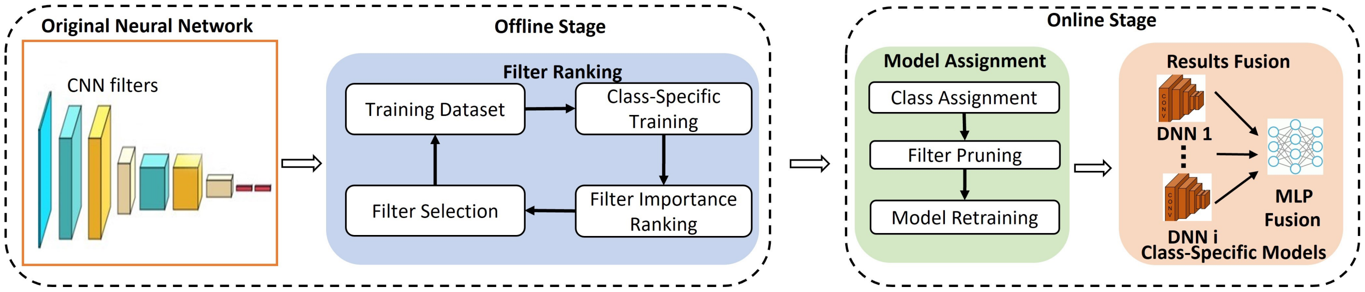


Fig. 2. Workflow of NNFacet to decompose a DNN model into multiple class-specific models for concurrent smart sensors.

the filter/neuron $k \in F \cup R$. Then, the OPT-1 is formally formulated as follows:

$$x^*_{ij}, y^*_{ik} = \underset{x_{ij}, y_{ik}, i \in S, j \in C, k \in F \cup R}{\arg\max} \; \min_{i \in S} \{E_i - L \cdot (e_{c,i} + e_{m,i})\}, \tag{4}$$

$$\text{s.t.} \quad L \cdot (e_{c,i} + e_{m,i}) \le E_i, \qquad \forall i \in S, \tag{5}$$

$$m_i \le M_i, \qquad \forall i \in S, \tag{6}$$

$$a_L \ge A_{\text{req}}, \tag{7}$$

$$\textstyle\sum_{i=1}^{|S|} x_{ij} = 1, \qquad \forall j \in C, \tag{8}$$

where m_i, $e_{c,i}$, and $e_{m,i}$ are given by Eqs. (1), (2), and (3), respectively. Eqs. (5), (6) and (7) ensure the energy and memory constraints as well as accuracy requirement, respectively. The constraint in Eq. (8) ensures that each class $j \in C$ is assigned to one model only.

Note that for the virtual concurrent sensors case discussed in §3.1.3, since all energy usages of the virtual concurrent sensors attribute to the smart camera, the objective function $\min_{i \in S}\{E_i - L \cdot (e_{c,i} + e_{m,i})\}$ should be updated to $E - \sum_{i \in S} L \cdot (e_{c,i} + e_{m,i})$, where E is the current remaining energy of the smart camera. The constraint Eq. (5) should be updated to $\sum_{i \in S} L \cdot (e_{c,i} + e_{m,i}) \le E$. In addition, M_i represents the smart camera's available RAM.

Solving OPT-1 faces the following practical challenges. First, although the inference accuracy a_L for any candidate solution may be measured with a validation dataset, the search process to solve OPT-1 may generate massive candidate models for all sensors, leading to extremely high computation overhead for the validation. Second, OPT-1 is an integer min-max optimization problem with an extremely large solution space. Considering the CNNs, just for the second dimension of the decision variable y_{ik} (i.e., k), the representative CNNs usually have thousands of filters and neurons. For instance, the VGG-19 variant used in the performance evaluation of this paper has a total of 5,504 convolutional filters and 2,048 FC neurons. Thus, the size of the search space of OPT-1 for just assigning filters/neurons to a certain sensor i sensitive to class j is 2^{7552}. Thus, OPT-1 must be reduced for tractability.

4 Design of NNFacet

In this section, we describe the main workflow of the proposed NNFacet to solve the OPT-1 problem for splitting CNN. Then, we present the details of three main steps of NNFacet. Finally, we extend NNFacet to address RNN.

4.1 Design Overview

To understand the complexity in solving OPT-1, we first describe a baseline method that solves OPT-1 without enforcing the accuracy constraint in Eq. (7). Specifically, we use MATLAB's genetic algorithm (GA) solver to solve OPT-1 without the accuracy constraint. Although GA is a generic method to solve complex optimization problems, due to the large number of filters and neurons, as shown shortly, the computation overhead of this baseline method is very high. Thus, this method is ill-suited for a resource-dynamic sensor system that requires a short assignment period.

To further reduce the computation overhead in solving OPT-1, we propose NNFacet that focuses on pruning the convolutional filters, as they are the most computation-intensive and account for most RAM usage. Under NNFacet, each class-specific model contains all the FC neurons of the original model. NNFacet reduces the solution search space by selecting the filters based on a heuristic but effective metric. Then, OPT-1 is reduced to an optimization problem with fewer decision variables, which can be solved in acceptable times.

Fig. 2 overviews the workflow of NNFacet, which consists of three main steps: *filter ranking*, *model assignment*, and *results fusion*. In the filter ranking step, NNFacet determines the importance of each filter in recognizing the input samples of a given class. Given the important filter sets of all classes, the solution search space of OPT-1 is significantly reduced. In the model assignment step, NNFacet finds the optimal class assignment. The class assignment problem is a complex optimization problem. Its solution space size exponentially increases with the numbers of classes and sensors. Beyond the generic GA solver, we also propose a greedy method to efficiently find the class assignment solution. With the class assignment, each class-specific model is formed by pruning the unimportant filters from the original CNN model and then retrained to compensate the accuracy loss caused by the pruning. In the last step, the results yielded by all class-specific models are fused by an MLP to generate the final inference result. In what follows, we describe the detailed designs of the above three steps.

4.2 Filter Ranking

This step aims to measure the importance of each filter of the original CNN to the learning classes using a ranking metric, called the average percentage of zero (APOZ). Specifically, in the CNNs, each convolutional layer is often followed by an activation layer that generates the activation map for the output of each fitter. The APOZ was proposed in [35] based on the observation that the activation map of a filter may consist of zeros activations which do not provide useful feature information to learn the output label of the model inputs. Thus, the filter with more zero activations is less important to the model learning accuracy. The experiments in [35] showed that the APOZ-based CNN compression approach can significantly reduce the size of two representative CNN models (i.e., LeNet and VGG-16) while maintaining the same accuracy, compared with the original models. Thus, in this study, we adopt this approach to form the class-specific models by pruning the less important filters in the original CNN.

To end this, we train the original CNN with a training dataset consisting of samples in all classes. Then, we sequentially determine the importance of the filters for recognizing every class. In particular, for a class j, we feed all training data samples of this class to the CNN and calculate the APOZ of all filters. Let $O^l_{f,j}$ denote the feature maps of the filter f in the convolutional layer l. Then, the APOZ value of the filter f in the layer l, denoted by $\mathrm{APOZ}^l_{f,j}$ is

$$\mathrm{APOZ}^l_{f,j} = \frac{\sum_{q=0}^{m} \sum_{p=0}^{h_o w_o} \Pi(O^l_{f,j}(q,p) = 0)}{m h_o w_o}, \tag{9}$$

where m is the number of input images; h_o and w_o represent the height and width of the activation map. $\Pi(x) = 1$ if x is true and $\Pi(x) = 0$ otherwise. The filter with a smaller APOZ value (i.e., fewer zeros in its output feature map) is more important in recognizing the class j. A filter is considered important for a class if its APOZ is lower than a threshold, denoted by ζ_{apoz}. The threshold ζ_{apoz} can be set to balance the trade-off between the model size and accuracy. A lower value of ζ_{apoz} leads to a smaller size of the class-specific model, but more accuracy loss caused by the filter pruning.

4.3 Model Assignment

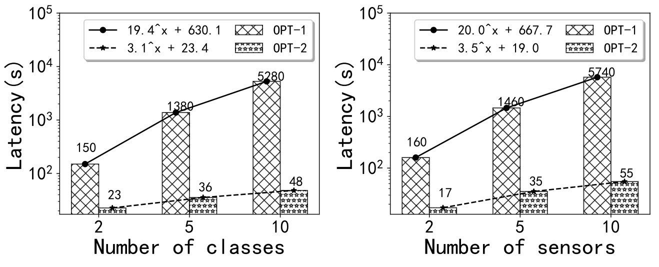
Given an APOZ threshold ζ_{apoz}, we can obtain the important filters assigned to the class-specific model for learning the class subset C_i. We will discuss the setting for ζ_{apoz} shortly. Thus, the model splitting problem OPT-1 can be reduced to the class assignment problem which focuses on assigning the classes to the class-specific models. In what follows, we present the optimization formulation of the class assignment problem and then a greedy algorithm to find the solution.

4.3.1 Optimization formulation

Given the filter assignment solution achieved in the filter ranking step, the OPT-1 is reduced to a class assignment problem, denoted by OPT-2, without the need of the decision variables y_{ik}. The OPT-2 is formulated as follows:

$$x^*_{ij} = \underset{x_{ij}, i\in S, j\in C}{\arg\max} \min_{i\in S}\{E_i - L\cdot(e_{c,i} + e_{m,i})\},$$
$$\text{s.t.} \quad \text{Eqs. (5), (6) and (8).}$$

OPT-2 does not include the inference accuracy requirement. It relies on the heuristic approach of APOZ-based filter selection to address the accuracy requirement. This relaxation exempts NNFacet from measuring the accuracy of the massive candidate class-specific models during the search



(a) Five sensors ($|S| = 5$). (b) Ten classes ($|C| = 10$).

Fig. 3. Latencies of solving reduced OPT-1 and OPT-2 using MATLAB's genetic algorithm (GA) solver and the curve fittings (y-axis is in log scale).

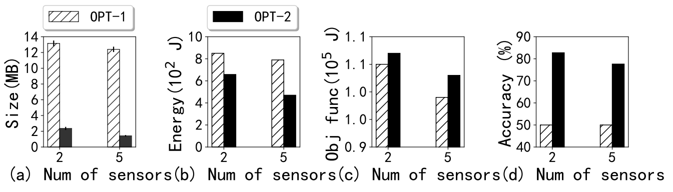


Fig. 4. Performance of model assignment solution in terms of (a) class-specific model size, (b) total energy usage of all class-specific models, (c) objective function value, (d) inference accuracy. The energy usage and accuracy results are determined based on 10,000 testing samples.

process of OPT-2. Compared to OPT-1, the number of decision variables of OPT-2 is reduced from $|S| \times (|C| + |F \cup R|)$ to $|S| \times |C|$. However, OPT-2 is NP-hard. A proof sketch is as follows. We consider a further relaxed problem where the residual energy E_i of all sensors is enough to process L samples and thus, the energy constraint (5) can be omitted. Then, OPT-2 can be mapped to a max-min 0-1 Knapsack (MMK) problem [36] that assigns a subset C_i of a set of items C (i.e., set of data classes) to a knapsack under $|S|$ scenarios (i.e., S sensors). The M_i is considered as the capacity of the knapsack under a scenario $i \in S$. Since the MMK problem is NP-hard [36] and OPT-2 is a more constrained problem, OPT-2 is also NP-hard.

Now, we analyze and evaluate the computation overheads for solving the relaxed OPT-1 (i.e., the baseline method in §4.1) and OPT-2 to split a representative CNN model called VGG-19 [1] into multiple class-specific models for learning CIFAR-10 [37] images. The numbers of sensors $|S|$ and data classes $|C|$ vary from 2 to 10. The residual energy E_i in Joule (J) and available memory M_i in megabyte (MB) of each sensor i are randomly selected from the ranges of [16, 80] MB and [10^5, 1.8×10^5] J, respectively. The settings for E_i and M_i will be explained in §5.1. For OPT-2, ζ_{apoz} is set to 80%. Both problems are solved using MATLAB's GA solver on a workstation computer with a 2.2GHz CPU and 32GB RAM. Fig. 3 shows the latency in solving the relaxed OPT-1 and OPT-2 under various settings of $|S|$ and $|C|$. The solving latencies of the two problems exponentially increase with $|C|$ and $|S|$. Moreover, due to OPT-2's fewer decision variables, the latencies in solving OPT-2 are significantly lower than those of OPT-1. For instance, from Fig. 3(b), when $|C| = 10$ and $|S| = 10$, the latency of the relaxed OPT-1 is about 1.6 hours, which is 104× longer than that of OPT-2. Fig. 4 shows the model size with standard deviation, total energy usage of running all class-specific models obtained by the feasible solutions of the relaxed OPT-1 and OPT-2,

Algorithm 1 A greedy algorithm for model assignment.

```
1:  Inputs: $\text{CNN}_0$ is the original model; $F$ is the set of filters in $\text{CNN}_0$; $E_i$ and $M_i$ are the remaining energy and available memory of sensor $i$; $\zeta_{\text{apoz}}^{\text{int}}$ is the initial value of the APOZ threshold $\zeta_{\text{apoz}}$; $\Delta\zeta_{\text{apoz}}$ is the AOPZ step size; $\Phi$ is the training dataset.
2:  $\zeta_{\text{apoz}} = \zeta_{\text{apoz}}^{\text{int}}$; $\Upsilon = True$;
3:  while $\Upsilon == True$ do
4:      $S = \{1, \ldots, |S|\}; C = \{1, \ldots, |C|\}$;
5:      $F_j^{\text{class}} = \textit{filRanking}(\text{DNN}_0, \Phi, \zeta_{\text{apoz}}), \forall j \in C$;
                                          ▷ important filters for class $j$.
6:      $F_i = \emptyset, \forall i \in S$;            ▷ filters assigned to model $i$.
7:      while $C \neq \emptyset$ do
8:          $i = \arg\max_{\forall i \in S} E_i$;
9:          if $F_i == \emptyset$ then
10:             $j = \arg\max_{\forall j \in C} |F_j^{\text{class}}|$;
11:         else
12:             $j = \arg\max_{\forall j \in C} |F_j^{\text{class}} \cap F_i|$;
13:         end if
14:         if $m_i(F_i \cup F_j^{\text{class}}) \leq M_i$ then
15:             $F_i = F_i \cup F_j^{\text{class}}$; $C = C \setminus \{j\}$;
16:             $E_i = E_i - e_i(F_i \cup F_j^{\text{class}})$;
17:         else
18:             $S = S \setminus \{i\}$;
19:             if $S == \emptyset$ then Break;
20:             end if
21:         end if
22:     end while
23:     if $C == \emptyset$ then         ▷ all classes in $C$ are assigned.
24:         $\text{CNN}_i = \textit{filPrune}(\text{CNN}_0, F_i), \forall i \in S$;
25:         $\text{CNN}_i = \textit{retrain}(\text{CNN}_i, \Phi), \forall i \in S$;
26:         $\Upsilon = \text{False}$;
27:     else
28:         $\zeta_{\text{apoz}} = \zeta_{\text{apoz}} - \Delta\zeta_{\text{apoz}}$;
29:     end if
30: end while
31: Return: $\text{CNN}_1, \ldots, \text{CNN}_{|S|}$ are class-specific models.
```

the objective function value, and the inference accuracy. The total energy usage and the accuracy achieved by fusing the results of the class-specific models are determined based on 10,000 testing images. The design of our fusion approach will be detailed in §4.4. From Figs. 3 and 4, NNFacet that solves OPT-2 can achieve better performance.

4.3.2 Greedy algorithm

From Fig. 3, the latency of solving OPT-2 using the GA solver increases exponentially with the numbers of sensors and data classes. To improve the scalability of NNFacet, we design a greedy algorithm as an alternative of the GA solver. Algorithm 1 presents the procedure of the greedy algorithm. Specifically, given sets of important filters, F_j^{class} ($j = 1, \ldots, |C|$) determined by the filter ranking step, the algorithm in lines 7-22 assigns the $|C|$ classes to at most $|S|$ models. To maximize the system lifetime, we first choose the sensor i with the highest residual energy E_i (i.e., $i = \arg\max_{\forall i \in S} E_i$) and assign it with the class j that has the highest number of important filters. Then, we assign the remaining classes to $|S|$ sensors in multiple iterations. In each iteration, the sensor with the highest E_i is chosen. If the sensor i has been assigned with some classes in previous iterations (i.e., $F_i \neq \emptyset$), among the remaining classes, we assign the class j that shares the maximum number of the same filters with the model i (i.e., $j = \arg\max_{\forall j \in C}(|F_j^{\text{class}} \cap F_i|)$) to the model i. The class j is assigned to the model i only when the size of model i with class j, denoted by $m_i(F_i \cup F_j^{\text{class}})$, is no greater than the sensor i's residual memory M_i. Then, the E_i is updated, i.e., $E_i = E_i - e_i(F_i \cup F_j^{\text{class}})$, where the $e_i(F_i \cup F_j^{\text{class}})$ denotes the per-sample energy usage of the model i assigned with the class j. Otherwise, the sensor i is considered memory-exhausted and excluded from the sensor set S in next assignment iterations. The iteration stops when all classes are assigned or there is no more available sensor, i.e., $S = \emptyset$.

A higher APOZ threshold ζ_{apoz} generally leads to higher inference accuracy but more memory and energy usages, because more filters are kept in the class-specific models. Thus, given a high ζ_{apoz}, the baseline method to solve the reduced OPT-1 and the greedy algorithm to solve OPT-2 may not find a feasible solution. Thus, we apply a greedy strategy to set ζ_{apoz}. Specifically, we initialize ζ_{apoz} to a high enough value denoted by $\zeta_{\text{apoz}}^{\text{int}}$ such that no feasible solution can be found and then gradually reduce ζ_{apoz} by a step size denoted by $\Delta\zeta_{\text{apoz}}$ until a feasible solution is found. The greedy algorithm implements this strategy by line 28.

When a feasible solution is found, we perform filter pruning denoted by *filPrune*(CNN_0, F_i) that removes all unimportant filters not in F_i from the CNN_0 to form the class-specific models CNN_i. The class-specific models are then retrained to compensate the accuracy loss caused by the pruning. To avoid the unbalanced training data problem, we pick the training samples of all classes assigned to the model CNN_i to form the first half of the retraining dataset. The second half with the same null-class label consists of the training samples from the remaining classes.

4.4 Results fusion

The results fusion step adopts a late fusion approach that aggregates the outputs from the penultimate layers before the classification layers of the class-specific models to yield the classification result. Specifically, we concatenate the output vectors of the penultimate layers of the class-specific models and feed the concatenated vector to an MLP to generate the final result. With a training dataset, this MLP can be trained once all the class-specific models are given.

4.5 NNFacet for Splitting RNNs

The detailed design presented in §4.2 and §4.3 is mainly for CNNs. As mentioned earlier, an RNN is often formed by multiple RNN units, in which the neurons of the neural networks are the most compute-intensive elements. Thus, for RNNs, NNFacet focuses on determining and retaining the important neurons of these neural network layers in the class-specific models, without considering filters. §5 will evaluate the effectiveness of NNFacet for RNNs.

TABLE 2
Description of datasets used in evaluation.

Application	Dataset	Description
Speech recognition	Google Speech Command [38]	105,000 one-second audio utterances of 35 keywords
	LibriSpeech [28]	1,000 hours of English speech corpus sampled at 16ksps
Vibration analysis	MaFaulDa [39]	1,951 5-second traces sampled by 8 vibration sensors at 50kHz
Video analytics[1]	CIFAR-10 [37]	60,000 color images of 10 object classess
	Caltech256 [40]	30,607 color images of 256 object classes
	MNIST [41]	70,000 grayscale images of handwritten digits

[1]We use series of static images to simulate the video traces.

5 Performance Evaluation

This section evaluates the effectiveness of NNFacet and the baseline approaches for the three applications discussed in §3.1. In what follows, we first present the experiment settings. Then, we present the evaluation results.

5.1 Evaluation Settings

For all three applications, we adopt VGG-19 [1] that consists of 16 convolutional layers with a total of 5,504 filters, 2 FC layers and 5 pooling layers. For each application, we apply NNFacet to decompose the VGG-19 into multiple class-specific models. To evaluate the applicability of NNFacet to RNNs, for the speech recognition application, we also design a bidirectional LSTM model that consists of two LSTM layers and a FC layer. The first and second LSTM layers have 1,024 and 128 neurons, respectively, while the FC layer has 128 rectified linear units (ReLUs). For LSTM, NNFacet assigns the important recurrent units in the LSTM layers to the class-specific LSTM models. For both the decomposed VGG-19 and LSTM models, we adopt a results fusion MLP that consists of an input layer, two hidden layers with 512 and 246 ReLUs respectively, and a softmax output layer.

We use TensorFlow 2.1 to build the above VGG-19 and LSTM models, and apply NNFacet implemented in Python 3.7 to decompose the models. In our performance evaluation, we mainly consider the Raspberry Pi-4B as the smart sensor platform. Note that we also consider the ESP32-CAM platform for the *in situ* video analytics application. For the Raspberry Pi platform, the setting of b in Eq. (1) for storing a 32-bit floating-point parameter is $b = 4\,$bytes. For the computing and memory access energy usage models in Eqs. (2) and (3), we set $e_{\text{mac}} = 1.48\,\text{nJ}$ and $e_{\text{mem}} = 139 \times e_{\text{mac}} = 205.72\,\text{nJ}$, respectively, according to the measurements in [32]–[34]. The residual energy E_i of each sensor i is randomly selected within a range from $10^5\,\text{J}$ to $1.8 \times 10^5\,\text{J}$, where the latter is the total energy of a Lithium battery with a capacity of $1000\,\text{mAh}$ at $5\,\text{V}$. Similarly, the sensor's remaining memory capacity M_i is randomly selected from a range of $[M_{\min}, M_{\max}]$. For the speech recognition application, we set $M_{\min} = 10\,\text{MB}$ and $M_{\max} = 20\,\text{MB}$. For the other two applications, we set $M_{\min} = 20\,\text{MB}$ and $M_{\max} = 40\,\text{MB}$. For all applications, the number of inference samples L is set to be 100. The initial APOZ thresholds $\zeta_{\text{apoz}}^{\text{int}}$ for the three applications are set to be 30%, 71%, and 81%, respectively. The decreasing step $\Delta\zeta_{\text{apoz}}$ of the APOZ threshold is set to 5%.

We employ five performance metrics: (1) *Remaining memory* is given by the sensor's memory capacity M_i minus the memory size of its class-specific model; (2) *Remaining energy* is the available energy of the sensor after running its class-specific model to process 100 data samples in one assignment period; (3) *System lifetime* is the time duration until the sensor with the minimum remaining energy runs out of its energy based on the assumption that the sensor remains in the active state and uses $5\,\text{W}$ baseline power; (4) *Latency* is the execution time of a class-specific model on a Raspberry Pi-4B for processing a sample; (5) *Accuracy* is the inference accuracy ratio of the testing samples.

We evaluate two variants of NNFacet called NNFacet-GA and NNFacet-greedy, which use MATLAB's GA solver and Algorithm 1 to solve OPT-2. We employ two baseline approaches:

■ **Cost-aware approach** minimizes the required number of sensors for the collaborative inference. To this end, the sensor with the maximum memory size is first selected and assigned with the maximum number of classes and the corresponding important filters to form a class-specific model that can fit into the sensor's memory. This process is repeated for the remaining sensors until all classes are assigned. Finally, the class-specific models are retrained to compensate the accuracy loss caused by the filter pruning.

■ **Memory-aware approach** proposed in our prior work [12] splits the original DNN model into multiple class-specific models such that the memory size of these models can be minimized. Specifically, the classes are assigned to the sensors in multiple iterations. In the first iteration, every sensor is assigned with at most one class. The sensor with the maximum memory size is assigned first with the class having the highest number of filters, and so on. Note that different classes may share some important filters. Thus, in the next iterations, the sensor is assigned with a remaining class that shares the maximum number of the same important filters with its current classes if its memory is not full. As such, the additional filters of each class-specific model (i.e., the model size) is minimized. The iteration stops when all classes are assigned.

We compare NNFacet with the above two baseline approaches for all three applications. For the video analytics, we also compare NNFacet with two CNN compression approaches proposed in [9], [10]. Moreover, we use six public datasets as summarized in Table 2 to evaluate the effectiveness of NNFacet and baseline approaches. The detailed description of each dataset is in next sections.

5.2 Application 1: Speech Recognition

We use the Google Speech Command V2 [38] and LibriSpeech [28] datasets to design DNNs for keyword spotting and automatic speech recognition, respectively. The Google Speech Command dataset consists of 105,000 one-second audio utterances of 35 keywords (e.g., "yes", "no", "one", and "two"). LibriSpeech contains about 1,000 hours of English speech corpus sampled at 16 ksps. For each audio sample, we compute the 40-dimensional Mel-Frequency

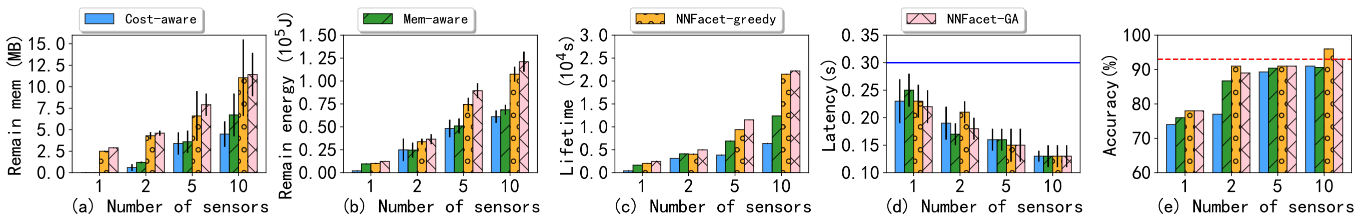


Fig. 5. Performance metrics of class-specific VGG-19 models on Google Speech Command dataset. Notes: In (a) and (b), error bar represents standard deviation of the metric over all class-specific models. In (d), error bar represents standard deviation of a class-specific model's execution times in 10 experiments. In (d) and (e), the solid and dotted lines represent the latency and accuracy of the original VGG-19 model, respectively. The above notes are also applicable to Figs. 6-12.

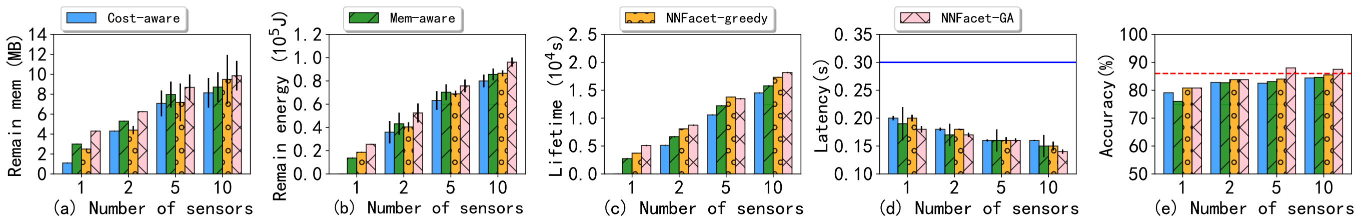


Fig. 6. Performance metrics of the class-specific VGG-19 models on LibriSpeech dataset.

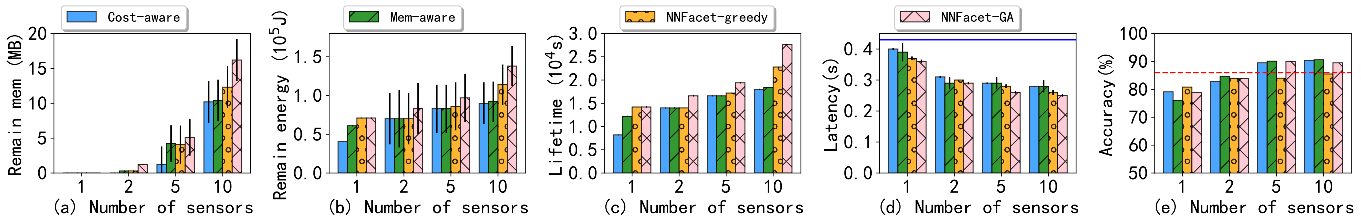


Fig. 7. Performance metrics of the class-specific LSTM models on Google Speech Command dataset.

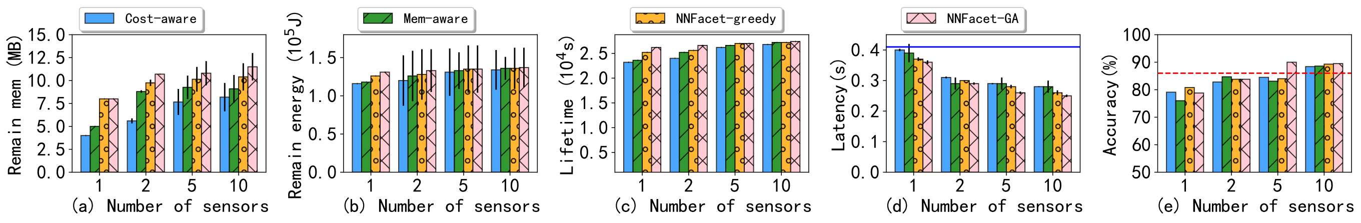


Fig. 8. Performance metrics of the class-specific LSTM models on LibriSpeech dataset.

Cepstral Coefficients (MFCC) frames. Eventually, a sample is converted to $40 \times 44 \times 1$ and $40 \times 50 \times 1$ MFCC tensors for the two datasets, respectively. For the experiments with these two datasets, we consider all classes, i.e., $|C| = 10$.

■ **Results for CNN-based speech recognition.** Figs. 5 and 6 show the remaining memory and energy, system lifetime, latency and accuracy of the VGG-19 class-specific models given by NNFacet-greedy and NNFacet-GA, as well as the baseline approaches on test samples of Google Speech Command and LibriSpeech datasets, respectively. We vary the number of sensors $|S|$ from 1 to 10. Note that when $|S| = 1$, the approaches only perform model compression by pruning filters and do not decompose the model. Absence of result in the figures means that no solution is found to meet the memory size constraint. The results show that, all the four performance metrics under any approach become better when $|S|$ increases. This is because, with more sensors, the class-specific model hosted by a single sensor generally needs to deal with fewer classes. As a result, the class-specific model can include more important filters/neurons for each class and achieve higher accuracy. From the results on remaining memory and energy, the class-specific models generated by NNFacet use less resources than those given by the baseline approaches. As a result, NNFacet extends the system lifetime. The class-specific models generated by the different approaches have similar latency and comparable/higher accuracy, compared with the models generated by the baseline approaches. Moreover, when $|S| \geq 2$, NNFacet achieves significantly lower latency while maintaining similar or even higher accuracy, compared with the original VGG-19 model. Compared with NNFacet-greedy, NNFacet-GA achieves better performance in terms of the memory, energy, system lifetime and latency while maintaining the similar accuracy. The main reason is that GA generally finds a solution closer to the optimal.

Table 3 shows the size of the original model and the total size of all class-specific models with $|S| = 5$. NNFacet-GA reduces the size by up to 13× compared with the original model and generates smaller models compared with the baseline approaches. This means that, even if we deploy all class-specific models on the same sensor, the memory and energy usages will be lower than those of the original model.

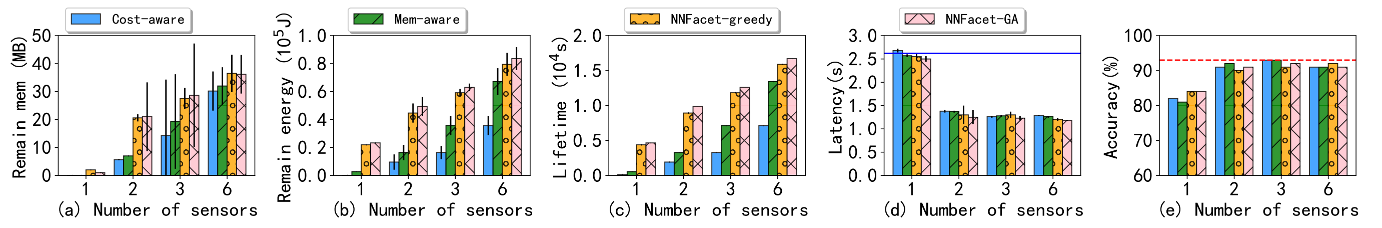


Fig. 9. Performance metrics of the class-specific VGG-19 models on microphone measurements of MaFaulDa dataset.

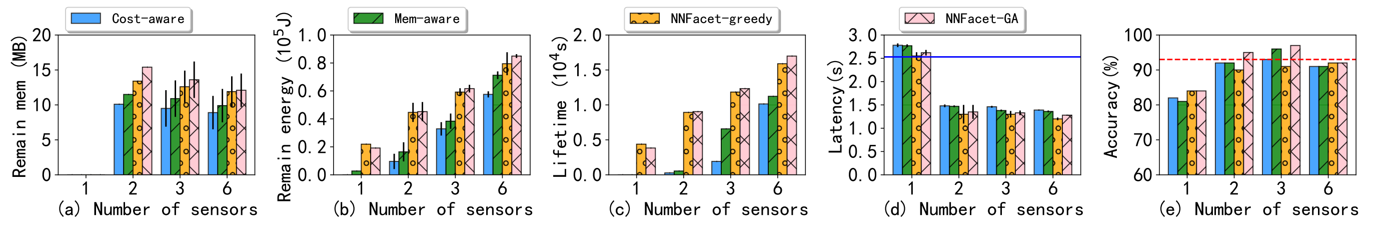


Fig. 10. Performance metrics of the class-specific VGG-19 models on tachometer measurements of MaFaulDa dataset.

TABLE 3
Size of the original model and total size of the class-specific VGG-19/LSTM models with $|S| = 5$.

Approach	Model Size (MB)					
	Speech recognition				Vibration analysis	
	VGG-19		LSTM		VGG-19	
	Goo[1]	Lib[2]	Goo[1]	Lib[2]	Mic[3]	Tac[3]
Original	241	261	128	142	864	841
Cost-aware	30.8	23.4	90.6	104	126	119
Memory-aware	29.6	21.4	84.5	96	100	97
NNFacet-GA	**22.8**	**19.5**	**77.0**	**83**	**77**	**65**

[1]Google Speech Command dataset for keyword spotting.
[2]LibriSpeech dataset for automatic speech recognition.
[3]Microphone & Tachometer data subsets of the MaFaulDa dataset.

However, from the result in Fig. 5(d), if these class-specific models are executed sequentially on the same sensor, the total latency will be higher than the latency of the original model. This is because the overhead of loading the model can outweigh the overhead of executing the model. However, for this application, the parallel execution of the class-specific models on the concurrent sensors ensures that the latency is shorter than that of the original VGG-19 model.

■ **Results for RNN-based speech recognition.** Figs. 7 and 8 present the performance metrics of all approaches for splitting the LSTM model into multiple class-specific models on the two datasets, respectively. The observations obtained for the CNN-based speech recognition are still applicable to the RNN-based speech recognition. Table 3 also shows the model size results when $|S| = 5$. NNFacet-GA achieves about 1.7× size reduction compared with the original model. All these results show that NNFacet remains effective for RNNs.

5.3 Application 2: Vibration Analysis

For the vibration analysis application, we use a machinery fault database (MaFaulDa) [39] that consists of 1,951 samples captured by eight vibration sensors, including six accelerometers, a tachometer, and a microphone attached on a machinery fault simulator. The eight sensors concurrently measure the vibration signals under six machinery states, i.e., normal function, inner and outer bearing faults, imbalance fault, horizontal and vertical misalignment faults. For each machinery state, each sensor captures eight traces, each of which is sampled at a rate of 50 kHz for five seconds. In our experiments, we use the microphone and tachometer data to classify the machinery state, respectively. We down-sample each sensor trace and arrange the result into a $224 \times 224 \times 1$ tensor, which is fed to the CNN model for classifying the machinery state. We divide the available samples to the training and testing sets by a ratio of 9:1.

Figs. 9 and 10 present the performance metrics of NNFacet and the baseline approaches on the microphone and tachometer datasets, respectively. Table 3 also includes the total size of all the class-specific models. The main observations from the results are: (1) NNFacet reduces the total size of all the class-specific models up to 13× compared with the original model; (2) NNFacet extends system lifetime by at least 35% compared with the baseline approaches; (3) NNFacet shortens the latency by 47% compared with the single-sensor system while achieving inference accuracy similar to the original model.

5.4 Application 3: *In Situ* Video Analytics

We use the CIFAR-10 [37], Caltech256 [40], and MNIST [41] image datasets to drive the evaluation. CIFAR-10 consists of 60,000 32×32 color images in 10 classes. We use 50,000 and 10,000 images for training and testing, respectively. Caltech256 consists of 30,607 images in 256 classes. The number of images in a class ranges from 80 to 827. We select a total of 3,706 images from 10 classes with the highest number of images for our evaluation. We use 2,964 and 742 images as the training and testing samples, respectively. We also select a total of 2,964 images from 3 classes to form another subset. These two data subsets are referred to as Cal10 and Cal3. MNIST is a handwritten digit dataset consisting of 70,000 28×28 grayscale images. We use 60,000 and 10,000 MNIST samples for training and testing. Note that as the three datasets consisting of static images can support evaluating the resource usages of the various approaches, we do not employ video traces to drive the evaluation.

In what follows, we first present the evaluation results of the class-specific models obtained by various approaches when deployed on Raspberry Pi-4B. Then, we

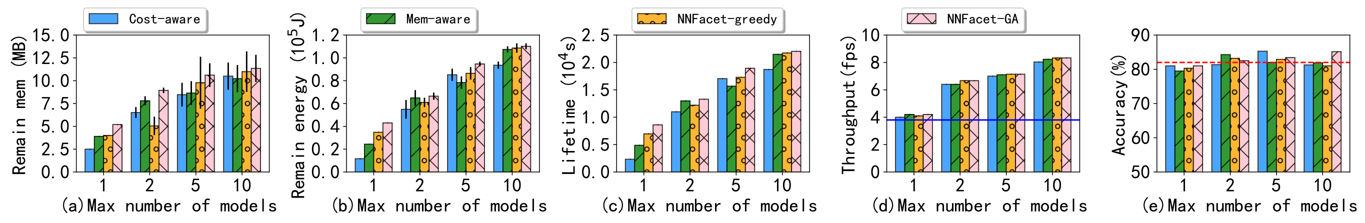


Fig. 11. Performance metrics of the class-specific VGG-19 models on CIFAR-10 dataset.

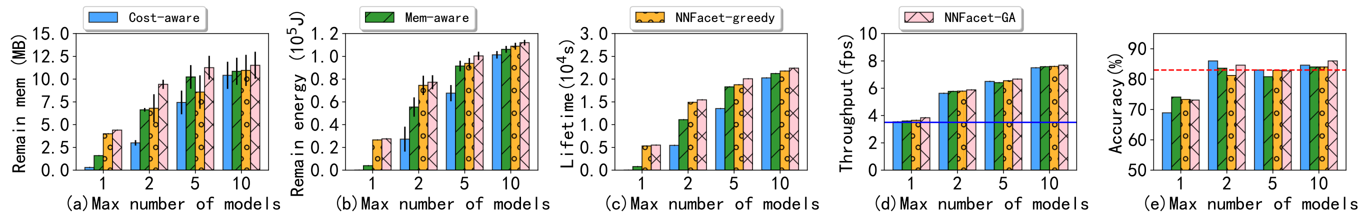


Fig. 12. Performance metrics of the class-specific VGG-19 models on Cal10 dataset.

TABLE 4
Comparison between NNFacet and baseline approaches including recent CNN compression techniques. (CIF refers to CIFAR-10)

Category	Approach	Model size (MB)			Accuracy (%)		
		CIF	Cal10	Cal3	CIF	Cal10	Cal3
Original VGG-19 model		241	241	240	83	82	97.7
Model splitting	Cost-aware	19.0	26	17.8	83.3	82	86
	Memory-aware	16.4	18.7	11.5	84.7	85.6	83.5
	NNFacet-GA	**14.1**	**13.1**	**9**	**85.5**	**84.6**	**83**
Model compression	DeepIoT [9]	2.9	–	–	40	–	–
	AppCNN [10]	–	–	50	–	–	98

TABLE 5
Performance of NNFacet-GA with MNIST on ESP32-CAM.

$\lvert S\rvert$	Max model size (KB)	Per-model latency (s)	Per-sample latency (s)	Accuracy[1] (%)
5	63	1.19	5.95	80.4
10	53	1.18	11.8	90.2

[1] Accuracy of the original VGG-19 model on MNIST is 98.3%.

evaluate NNFacet-GA on the more memory-constrained ESP32-CAM.

5.4.1 Evaluation on Raspberry Pi-4B

Figs. 11 and 12 show the performance metrics of NNFacet variants and the baseline approaches with the VGG-19 model on the Cal10 and CIFAR-10 testing images. In this video analytics application, we measure the image processing throughput in terms of frames per second (fps), which is more meaningful than the latency as measured in previous applications. From the results, NNFacet variants extend the system lifetime, compared with the two baseline approaches. Moreover, the class-specific models obtained by all approaches have higher processing throughput and accuracy than the original VGG-19 model, when $|S| \geq 2$. We also observe that, when $|S| \geq 2$, the left memory/energy and system lifetime decrease with $|S|$. The reasons are two-fold. First, in this application, all virtual concurrent sensors' class-specific models run on the same sensor. Second, the total size of all class-specific models increases with $|S|$. Nevertheless, on the CIFAR-10 dataset, the system lifetime under NNFacet when $2 \leq |S| \leq 5$ is longer than that when $|S| = 1$. On the Cal10 dataset, the system lifetime under NNFacet when $2 \leq |S| \leq 10$ is longer than that when $|S| = 1$.

We also compare NNFacet-GA with several existing CNN compression approaches as follows. First, DeepIoT [9] adopts the dropout operations to prune a maximum number of redundant filters in the convolutional layers while maintaining the same accuracy as that of the original CNN. Second, AppCNN [10] generates an approximated CNN of the original CNN by keeping a few front convolutional layers only. It also replaces the FC and softmax layers by a trained non-neuron network classifier. Table 4 shows the total model memory size and accuracy achieved by the original VGG-19 model and all approaches on the CIFAR-10 and Cal10 datasets. Compared with the cost-aware and memory-aware model splitting approaches, NNFacet-GA achieves the smallest total model size and the highest accuracy on the CIFAR-10 dataset. On the Cal10 dataset, NNFacet-GA outperforms the cost-aware approach in terms of both model size and accuracy. Although the accuracy of NNFacet-GA is 1% lower than that of the memory-aware approach, the total model size of NNFacet is about 30% smaller than that of the memory-aware approach. As reported in [9], although DeepIoT can compress the model for CIFAR-10 to 2.9 MB, the compressed model can only achieve 40% accuracy. Differently, NNFacet-GA achieves 85.5% accuracy with 14.1 MB total model size. For AppCNN, Table 4 includes the result from [10] where AppCNN achieves the minimum model size. From Table 4, we can see that the AppCNN can achieve higher accuracy than the proposed NNFacet. However, the size of the compressed VGG-19 model obtained by the AppCNN is 3x higher than that of the NNFacet-GA. In contrast, our NNFacet-GA approach always achieves the satisfactory trade-off between the mode size reduction and the accuracy.

5.4.2 Evaluation on ESP32-CAM

ESP32-CAM has only 512 KB static RAM. This memory is further divided to two parts, 200 KB for instructions and 320 KB for data. The DNN and its activation need to fit into the 320 KB RAM. Different from the Raspberry Pi-4B that has large enough RAM to store all the class-specific models, on ESP32-CAM, the models need to be loaded to RAM sequentially to process consecutive image frames. NNFacet-GA can find feasible solutions when $|S| \geq 5$. As shown in Table 5, the model size of a single class-specific model is 53KB and 63KB, when $|S|$ is 5 and 10, respectively. The time needed to run a single model on ESP32-CAM is 1.18 to 1.19 seconds. As a result, when $|S|$ is 5 and 10, the total time needed to classify a sample is 5.95 s and 11.8 s, respectively. On MNIST, the accuracy on the testing samples is 80.4% and 90.2% when $|S|$ is 5 and 10, respectively. Note that the original VGG-19 model achieves an accuracy of 98.3% but cannot fit into the RAM of ESP32-CAM. From the results shown in Table 5, NNFacet enables running a DNN on a single highly RAM-constrained smart sensor and provides a knob for the system designer to trade off the latency and accuracy in inferring a sample.

6 Conclusion and Future Work

This paper designed NNFacet, a DNN model splitting approach that allows executing the DNN-based advanced data analytics on concurrent resource-constrained sensors. We formulated an optimization problem with the objective of splitting the large DNN into multiple lightweight class-specific models that maximize the system lifetime and accuracy while satisfying the sensor′ memory and energy constraints. We developed solutions to the formulated problem. Extensive evaluation on three case studies and comparisons with four baseline approaches show the superior effectiveness of NNFacet. Specifically, NNFacet achieves longer system lifetime and more energy and memory savings, while maintaining accuracy and latency, compared to the baseline approaches.

As future work, it is interesting to extend NNFacet to address other DNN-based tasks such as semantic image segmentation which aims to classify each pixel of an input image into a particular class of an object. Specifically, NNFacet can be used to split a large multi-object segmentation DNN model into multiple object-specific models, each of which is responsible for generating a pixel mask that segments out areas of a subset of objects from the image input. Filter ranking and object assignment methods similar to those presented in this paper can be applied to form the object-specific models. Finally, instead of using MLP to perform late fusion, the output masks of the object-specific models can be simply merged to yield the final image segmentation mask.

Acknowledgment

This study is supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner, HP Inc., through the HP-NTU Digital Manufacturing Corporate Lab.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, May 7-9, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.

[3] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *ICMR*, 2016, pp. 173–182.

[4] Y. Zhang, T. Gu, and X. Zhang, "MDLdroidLite: a release-and-inhibit control approach to resource-efficient deep neural networks on mobile devices," *IEEE Transactions on Mobile Computing*, 2021.

[5] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NIPS*, 2014, p. 1269–1277.

[6] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[7] R. Shi, P. Dong, T. Geng, Y. Ding, X. Ma, H. K.-H. So, M. Herbordt, A. Li, and Y. Wang, "CSB-RNN: A faster-than-realtime rnn acceleration framework with compressed structured blocks," in *ACM ICS*, 2020, pp. 1–12.

[8] C. Chatzikonstantinou, D. Konstantinidis, K. Dimitropoulos, and P. Daras, "Recurrent neural network pruning using dynamical systems and iterative fine-tuning," *Neural Networks*, vol. 143, 2021.

[9] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *ACM SenSys*, 2017.

[10] C. Alippi, S. Disabato, and M. Roveri, "Moving convolutional neural networks to embedded systems: the AlexNet and VGG-16 case," in *ACM/IEEE IPSN*, 2018, pp. 212–223.

[11] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *ACM MobiCom*, 2018, pp. 115–127.

[12] C. Jiale, D. Van Le, T. Rui, and H. Daren, "Split convolutional neural networks for distributed inference on concurrent iot sensors," in *IEEE ICPADS*, 2021.

[13] G. Wang, Z. Liu, S. Zhuang, B. Hsieh, J. Gonzalez, and I. Stoica, "Sensai: Fast convnets serving on live data via class parallelism," in *MLOps Systems workshop in MLSys*, 2020.

[14] S. Tuli, G. Casale, and N. R. Jennings, "Splitplace: AI augmented splitting and placement of large-scale neural networks in mobile edge environments," *IEEE Transactions on Mobile Computing*, 2022.

[15] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in iot," *IEEE Internet of Things Journal*, vol. 7, 2020.

[16] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Interspeech*, 2015.

[17] V. Passricha and R. K. Aggarwal, *Convolutional neural networks for raw speech recognition*. IntechOpen, 2018.

[18] S. Doebling, C. Farrar, M. Prime, and D. Shevitz, "Damage identification and health monitoring of structural and mechanical systems from changes in their vibration characteristics: A literature review," *Technical Report*, no. LA-13070-MS, 1996.

[19] A. Bonde, S. Pan, M. Mirshekari, C. Ruiz, H. Y. Noh, and P. Zhang, "Oac: Overlapping office activity classification through iot-sensed structural vibration," in *ACM/IEEE IoTDI*, 2020, pp. 216–222.

[20] J. Zhou, X. Yang, L. Zhang, S. Shao, and G. Bian, "Multisignal vgg19 network with transposed convolution for rotating machinery fault diagnosis based on deep transfer learning," *Shock and Vibration*, 2020.

[21] L. Wen, X. Li, L. Gao, and Y. Zhang, "A new convolutional neural network-based data-driven fault diagnosis method," *IEEE Transactions on Industrial Electronics*, vol. 65, pp. 5990–5998, 2017.

[22] A. Zollanvari, K. Kunanbayev, S. A. Bitaghsir, and M. Bagheri, "Transformer fault prognosis using deep recurrent neural network over vibration signals," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–11, 2020.

[23] "The world's smallest autonomous racing drone," https://robohub.org/the-worlds-smallest-autonomous-racing-drone/, 2019.

[24] V. Iyer, A. Najafi, J. James, S. Fuller, and S. Gollakota, "Wireless steerable vision for live insects and insect-scale robots," *Science robotics*, vol. 5, no. 44, p. eabb0839, 2020.

[25] A. Mathur, A. Isopoussu, F. Kawsar, N. Berthouze, and N. D. Lane, "Mic2mic: using cycle-consistent generative adversarial networks to overcome microphone variability in speech systems," in *Proceedings of the 18th international conference on information processing in sensor networks*, 2019, pp. 169–180.

[26] W. Luo, Z. Yan, Q. Song, and R. Tan, "Phyaug: Physics-directed data augmentation for deep sensing model transfer in cyber-physical systems," in *ACM/IEEE IPSN*, 2021, pp. 31–46.
[27] Y. Miao and F. Metze, "Distance-aware dnns for robust speech recognition," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.
[28] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *IEEE ICASSP*, 2015.
[29] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *SIGCOMM'18*.
[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *IEEE CVPR*, 2016, pp. 2818–2826.
[31] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *51st asilomar conference on signals, systems, and computers*, 2017, pp. 1921–1925.
[32] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016.
[33] "The GFLOPS/W of the various machines in the VMW Research Group," https://bit.ly/3DJDDat.
[34] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.
[35] H. Hu, R. Peng, Y. Tai, and C. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *CoRR*, vol. abs/1607.03250, 2016.
[36] G. Yu, "On the max-min 0-1 knapsack problem with robust optimization applications," *Operations Research*, vol. 44, no. 2, p. 407–415, apr 1996.
[37] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images." Citeseer, 2009.
[38] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *CoRR*, vol. abs/1804.03209, 2018.
[39] "Machinery fault database," https://bit.ly/3l3D6cj, Online.
[40] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset." California Institute of Technology, 2007.
[41] "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/.

Jiale Chen received the BEng degree in optoelectronics information science and engineering from South China University of Technology, in 2018 and the MS in electronics engineering from Hong Kong University of Science and Technology, in 2019. He is currently working toward the PhD degree at the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU), Singapore. He is also working as a research associate with HP-NTU Corp Lab, NTU. His research focuses on efficient designs of deep learning models for IoT objects.

Duc Van Le is a Research Fellow at School of Computer and Engineering, Nanyang Technological University, Singapore. Previously, he was a Research Fellow (2016-2018) at Department of Computer Science, National University of Singapore. He received the Ph.D. (2016) degree in computer engineering from University of Ulsan, South Korea, and the B.Eng. (2011) degree in electronics and telecommunications engineering from Le Quy Don Technical University, Vietnam. His research interests include sensor networks, IoT sensing and computing in cyber-physical systems. He is a Senior Member of IEEE.

Rui Tan is an Associate Professor at School of Computer Science and Engineering, Nanyang Technological University, Singapore. Previously, he was a Research Scientist (2012-2015) and a Senior Research Scientist (2015) at Advanced Digital Sciences Center, a Singapore-based research center of University of Illinois at Urbana-Champaign, and a postdoctoral Research Associate (2010-2012) at Michigan State University. He received the Ph.D. (2010) degree in computer science from City University of Hong Kong, the B.S. (2004) and M.S. (2007) degrees from Shanghai Jiao Tong University. His research interests include cyber-physical systems, sensor networks, and pervasive computing systems. He is the recipient of ICCPS'22 Best Paper Award Finalist, SenSys'21 Best Paper Award Runner-Up, IPSN'21 Best Artifact Award Runner-Up, IPSN'17 and CPSR-SG'17 Best Paper Awards, IPSN'14 Best Paper Award Runner-Up, PerCom'13 Mark Weiser Best Paper Award Finalist, and CityU Outstanding Academic Performance Award. He is currently serving as an Associate Editor of the ACM Transactions on Sensor Networks. He also serves frequently on the technical program committees (TPCs) of various international conferences related to his research areas, such as SenSys, IPSN, and IoTDI. He is the TPC Co-Chair of e-Energy'23. He received the Distinguished TPC Member recognition thrice from INFOCOM in 2017, 2020, and 2022. He is a Senior Member of IEEE.

Daren Ho received the BS degree from School of EEE, Nanyang Technological University, in 2000, and the MS degree in digital manufacturing from Institute of System Science, National University of Singapore, in 2018. Currently, he is a Principal Engineer at HP Inc. with strong interest in digital factory transformation. He is holding up a few projects within HP-NTU Corp Lab working closely with NTU professors and research staffs to introduce Industrial 4.0 to HP factory in the area of vision inspection for product quality assessment.