Split Convolution Neural Networks for Distributed Inference on Concurrent IoT Sensors

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

***Abstract*—Convolutional neural networks (CNNs) are increasingly adopted on resource-constrained sensors for *in-situ* data analytics in Internet of Things (IoT) applications. This paper presents a model split framework, namely, splitCNN, in order to run a large CNN on a collection of concurrent IoT sensors. Specifically, we adopt CNN filter pruning techniques to split the large CNN into multiple small-size models, each of which is only sensitive to a certain number of data classes. These class-specific models are deployed onto the resource-constrained concurrent sensors which collaboratively perform distributed CNN inference on their same/similar sensing data. The outputs of multiple models are then fused to yield the global inference result. We apply splitCNN to three case studies with different sensing modalities, which include the human voice, industrial vibration signal, and visual sensing data. Extensive evaluation shows the effectiveness of the proposed splitCNN. In particular, the splitCNN achieves significant reduction in the model size and inference time while maintaining similar accuracy, compared with the original CNN model for all three case studies.**

***Index Terms*—Distributed CNN Inference; Speech Recognition; Vibration Analysis; Video Analytics.**

I. Introduction

The convolutional neural networks (CNNs) have been increasingly employed in Internet of Things (IoT) applications. Among numerous deep learning (DL) models, the representative CNN models such as VGGNet [1], ResNet [2] and GoogLeNet [3] are state-of-the-art techniques for IoT applications such as the video analytics and speech recognition. However, due to their complex configurations, the execution of such advanced CNN models often incurs high computing overhead and memory usage. For instance, the VGG-19 consists of more than 21 million parameters and requires about 241MB memory. Thus, the inference of the CNN models is traditionally conducted on high performance computing devices. Specifically, the sensing data (e.g., video and audio) are transmitted from the distributed sensors to a centralized cloud server or a fog node, where the CNN-based data processing is executed. However, this offloading method often suffers from several issues including high transmission latency and communication bandwidth usage, privacy concerns, and poor scalability, especially when the wireless communication is adopted for a cordless setting. As a result, the real-time data analytics that is critical to delay-sensitive applications (e.g., industrial monitoring and inspection) may not be achieved due to insufficient bandwidth and high communication latency.

To address the above challenges, we propose a model split framework, called splitCNN, which leverages the filter CNN compression techniques to decompose the multi-class CNN into multiple small-size models, each of which is sensitive to a certain number of object classes only. The class-specific models are deployed on resource-constrained concurrent IoT sensors which can simultaneously obtain the same or similar sensing data. These sensors collaboratively perform the distributed CNN inference by running their class-specific models, which enables the on-device advanced data analystics with low latencies. Specifically, given a large multi-class CNN model, our proposed splitCNN framework begins with determining the importance of all filters in the convolutional layers for learning each class. Then, we design an assignment algorithm that uses rankings of filters to assign all classes to multiple clusters such that the original CNN model is decomposed into an appropriate number of small-size models, each of which can fit in an IoT sensor with a certain memory capacity. Finally, we adopt a late fusion approach to fuse the outputs of multiple class-specific models to yield the final result.

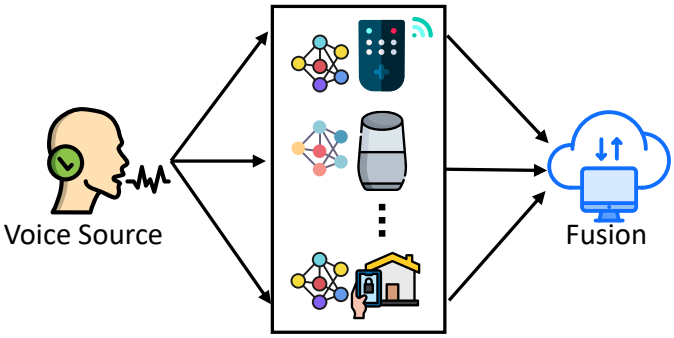


Fig. 1. An illustration of the splitCNN for collaborative speech recognition on concurrent voice sensing devices. Each device runs a small-size class-specific CNN model that is only sensitive to a few voice classes. The outputs of multiple devices are fused to yield the final speech recognition result.

Many studies [4]–[10] have proposed various CNN compression techniques which can reduce the memory size and computation cost of a large CNN model, such that the compressed model can fit in the resource-constrained IoT sensors. These existing studies can be divided into two categories which are model parameter and filter pruning. The parameter pruning approaches [4]–[6] focused on reducing the model size by pruning the redundant model parameters. However, merely pruning the model parameters may not lead to significant reduction in the computation overhead which is proportional to the inference latency. On the other hand, the filter pruning can reduce both the model size and computation overhead sig-

nificantly [7]–[10]. Thus, we adopt a filter pruning technique to generate the small-size models from a large CNN model.

However, the above existing studies only focused on compressing the multi-class CNN model into one model with reduced size and computation overhead. Since the model size of a trained CNN model increases with the number of training classes, one compressed model with many classes may still not fit in the resource-constrained sensor. Thus, our proposed approach splits the multi-class CNN model into class-specific models, each of which is responsible for learning a subset of the classes. We aim at assigning an appropriate number of classes to each model, such that it can be always deployed on an IoT sensor with a certain and limited memory capacity.

We apply the proposed splitCNN framework to three case studies with different sensing tasks. The first case study aims at performing the collaborative CNN-based speech recognition (e.g., keyword spotting (KWS) and automatic speech recognition (ASR)) on multiple concurrent human voice sensing devices. As illustrated in Fig. 1, the original model is decomposed into multiple class-specific models deployed on IoT sensors (e.g., smart home devices and smartphones). Given a voice sample, each sensor feeds its measured data into its model. The outputs of all sensors are fused for the speech recognition at a centralized node. The second case study is the vibration analysis, which is important for predictive maintenance in industrial systems [11]. We propose to deploy class-specific models on multiple vibration sensors that measure the vibration signal from the same vibrating object (e.g., motor). Each sensor runs its class-specific model to process its measured vibration signal, and then sends the model output to a centralized unit for result fusion.

The third case study aims at performing the low-power video analytics on wireless cameras. Different from the first and second case studies that use multiple concurrent sensors, this one considers deploying all class-specific models in one wireless camera to execute the CNN inference for video analytics. At runtime, the camera runs these small-size models on consecutive image frames with the similar contents, where each model is fed with a frame. Then, the model outputs on these consecutive frames are fused to determine whether the interested objects appear in the camera's field-of-view. Our approach avoids transmitting the video data from the camera to a remote node for advanced image processing. The evaluation results for these three case studies show that the splitCNN approach can always achieve significant reduction in the model memory usage and the latency of the advanced analytics, while achieving the high accuracy of the original CNN model.

The remainder of this paper is organized as follows. §II reviews related work. §III describes the design of the proposed splitCNN. §IV presents three case studies. §V presents evaluation results. §VI concludes this paper.

II. Related Work

A number of studies [4]–[10] have proposed various CNN compression approaches that allow running deep CNN models on embedded devices with limited computing resources. The studies in [4]–[6] focused on pruning the model parameters (e.g., weights) to reduce the memory required to store and run the deep CNN models. For instance, Han *et al.* [4] compressed a CNN model by removing the redundant connections with small weights. Then, the 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. Denton *et al.* [6] applied singular value decomposition to compress convolutional layers, and then fine-tuned these approximated layers to restore the accuracy. Those parameter pruning approaches can reduce the number of parameters in the fully connected layers. However, they often fall short of reducing the parameters in the convolutional layers which are the most compute-intensive layers.

The studies in [7]–[10] focused on pruning the filters that play the role of the feature extractors in the convolutional layers. Both the size and computation load of the CNNs can be significantly reduced by pruning these filters. For instance, Fang *et al.* [10] 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 less important filters and retrains the pruned model to compensate the accuracy loss caused by the pruning. The iteration stops when the pruned CNN cannot provide the minimum accuracy required by the designer. Yao *et al.* [7] 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.* [8] 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., the feed-forward neural network, vector machine or decision tree). A filter selection mechanism was proposed to further reduce the computation load by removing the less important filters.

Our work is inspired by the study in [12] that has shown the feasibility of decomposing a multi-class CNN model into multiple binary models, each of which consists of the important neurons for a specific class only. Similar to [12], we adopt a filter pruning technique [13] to decompose a CNN model into multiple class-specific models such that the high-accuracy and low-latency CNN-based applications can be achieved on the resource-constrained IoT devices. However, the approach in [12] only supports decomposing the N-class CNN model into N binary models. Beyond that, we propose an algorithm that assigns N classes into an appropriate number of the M-class models, where $1 \leq M \leq N$, such that the number of required devices is minimized while meeting the constraint on the model size. Moreover, we also consider a late fusion approach to fuse the outputs of the class-specific models to yield the final inference result.

III. Design of The Proposed Approach

A. Approach Overview

In general, a CNN model is formed by four types of layers: convolutional, pooling, activation and fully connected layers

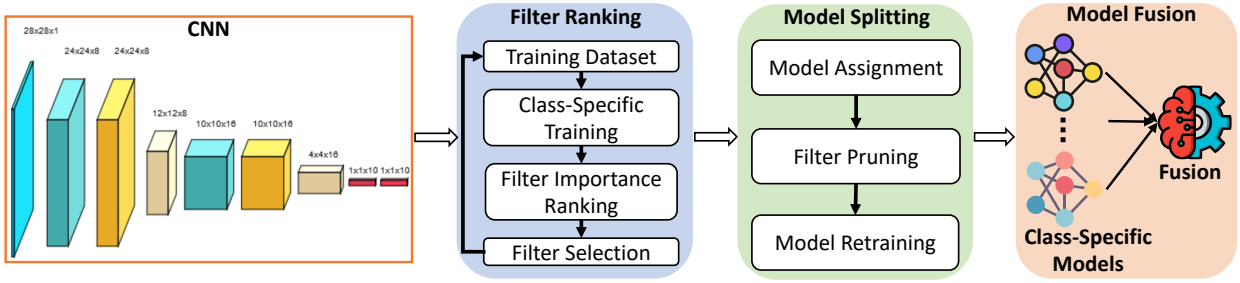


Fig. 2. Workflow of splitCNN to decompose a CNN model into multiple class-specific models for the IoT sensors.

among which the convolutional layers are the most compute-intensive. In particular, the convolutional layer contains a set of 3D filters which play the roles of extracting invariant local two-dimensional features. Pruning the filters helps reduce both the model parameters and computation overhead [10], [14]. Each 3D filter admits the feature maps generated by filters in the previous convolutional layer inputs to extract an output feature map that are then fed into the following convolutional layer for further feature extraction. Let m_{l-1}, w_{l-1} and h_{l-1} denote the number of input feature maps, the width, and height of each input feature map in the convolutional layer i, respectively. Given the input feature maps m_{l-1}, each filter in the layer l that is composed of m_{l-1} 2D kernels $\mathcal{K} \in R^{k\times k}$ generates one output feature map. Thus, pruning one filter f_l in the layer i reduces $m_{l-1}k^2$ parameters and $m_{l-1}k^2w_lh_l$ floating point operations (FLOPs). In the next convolutional layer $l+1$, the m_{l+1} 2D kernels applied on the output feature map generated by the pruned filter f_l are also removed, which results in additional reduction of k^2m_{l+1} parameters and $k^2m_{l+1}w_{l+1}h_{l+1}$ FLOPs. In summary, pruning a filter can lead to the total reduction of $k^2(m_{l-1}+m_{l+1})$ parameters and $k^2(m_{l-1}w_lh_l + m_{l+1}w_{l+1}h_{l+1})$ FLOPs.

Our proposed splitCNN aims at pruning the less important filters from the original multi-class CNN model to generate the small-size class-specific models, each of which is sensitive to a certain number of classes. Fig. 2 overviews the workflow of splitCNN, which consists of three main steps: filter ranking, model splitting, and model fusion. In particular, the splitCNN begins with the filter ranking to determine the importance of each filter in the convolutional layers in recognizing the specific classes in the training dataset. The model splitting step aims at assigning the classes into the appropriate number of class-specific models, such that the constraint on memory usage can be satisfied. In the last step, the spitCNN fuses the results from multiple class-specific models obtained by the previous steps to yield the final classification result.

B. Workflow of splitCNN

In what follows, we detail how the proposed splitCNN decomposes a multi-class CNN model into class-specific models to be deployed on the IoT sensors.

Filter Ranking: Assume that we have a training dataset consisting of N classes. We first train the original CNN using the entire training dataset. Then, we sequentially determine the importance of filters (i.e., neurons) in the convolutional layers for recognizing every class i. In particular, for a class i, we feed all training data samples of the class to the CNN. Then, we adopt the average percentage of zero (APOZ) [13] to measure the importance of the filters. In the CNN model, each convolutional layer is often followed by an activation layer for creating the feature maps. The APOZ is defined as the percentage of zero activation of a filter after the activation mapping. Let O_c^l denote the output of the channel c in the convolutional layer l. Then, the APOZ_f^l of the filter f in the layer l is calculated as

$$\text{APOZ}_f^l = \text{APOZ}(O_c^l) = \frac{\sum_k^M \sum_j^D F(O_{c,j}^l(k)=0)}{M \times D}, \quad (1)$$

where M is the number of input images, D is the dimension of the activation map of O_c^l, $F(.)=1$ if true and $F(.)=0$ if false. The filter with a smaller APOZ value (i.e., fewer zeros in its output feature map) is more important in recognizing the class i. Thus, we rank the importance of each filter for each class according to the corresponding AOPZ value. In particular, a filter is considered important for a specific class if its AOPZ is lower than a threshold. We use a different APOZ threshold to prune the less important filters on the different convolutional layers. For a specific layer, the threshold is selected to balance the trade-off between the model size and accuracy. A lower threshold leads to a smaller model size but more accuracy loss caused by the filter pruning. Let $\mathcal{F}_i$ denote a set of important filters for the class i.

Model Splitting: We assume that the IoT sensor has a memory capacity of Φ_{th}, which is not sufficient to run the original CNN model. This step aims at assigning N classes into an appropriate number of class-specific models such that the memory constraint can be satisfied. Let K denote the number of class-specific models. We consider the following application scenarios. In Scenario A, the class-specific models are deployed on concurrent sensors, each of which runs a model to process the same data input. In this scenario, our goal is to decompose the original CNN model into the minimum number of models, while the largest model size is less than the sensor's memory capacity of Φ_{th}. As such, the minimum number of sensors is required to perform the collaborative inference. In Scenario B, all the class-specific models are deployed on a single sensor. The number of models is determined such that the total size of all class-specific models is less than the sensor's memory capacity of Φ_{th}.

Algorithm 1 Assign N classes into K models.

1: **Inputs:** CNN_0 is original model; $\mathcal{F}_i$ is set important filters for class i; Φ_c is sensor's memory capacity.
2: $K = \lceil \frac{N}{2} \rceil$; $\Upsilon = \text{True}$;
3: **while** $\Upsilon == \text{True}$ & $1 \leq K \leq N$ **do**
4: $\quad \mathcal{C}_k = \emptyset \ \forall k = 1, \ldots, K$; ▷ set of filters in model k.
5: $\quad \mathcal{N} = \{1, \ldots, N\}$; ▷ set of N classes.
6: $\quad$ **for** each $k = 1, \ldots, K$ **do**
7: $\quad\quad j = \arg\max_{\forall i \in \mathcal{N}} |\mathcal{F}_i|$
8: $\quad\quad \mathcal{C}_k = \mathcal{C}_k \cup \mathcal{F}_j$; $\mathcal{N} = \mathcal{N} \setminus \{j\}$;
9: $\quad$ **end for**
10: $\quad$ **while** $\mathcal{N} \neq \emptyset$ **do**
11: $\quad\quad k = \arg\min_{\forall i = 1, \ldots, K} |\mathcal{C}_i)|$;
12: $\quad\quad j = \arg\min_{\forall i \in \mathcal{N}} (\mathcal{F}_i \cap \mathcal{C}_k)$;
13: $\quad\quad \mathcal{C}_k = \mathcal{C}_k \cup \mathcal{F}_j$; $\mathcal{N} = \mathcal{N} \setminus \{j\}$;
14: $\quad$ **end while**
15: $\quad$ **for** each $k = 1, \ldots, K$ **do**
16: $\quad\quad \text{CNN}_k = \textit{filPrune}(\text{CNN}_0, \mathcal{C}_k)$;
17: $\quad\quad \text{CNN}_k = \textit{retrain}(\text{CNN}_k)$;
18: $\quad$ **end for**
19: $\quad$ **if** A **then** $\Phi_m = \max(\Phi_k)$;
20: $\quad$ **else if** B **then** $\Phi_m = \sum_{k=1}^{k=K} \Phi_k$;
21: $\quad$ **end if**
22: $\quad$ **if** $\Phi_m < \Phi_{\text{th}} - \delta$ **then**
23: $\quad\quad K = K - 1$;
24: $\quad$ **else if** $\Phi_m > \Phi_{\text{th}}$ **then**
25: $\quad\quad K = K + 1$;
26: $\quad$ **else**
27: $\quad\quad \Upsilon = \text{False}$;
28: $\quad$ **end if**
29: **end while**
30: **Return**: $\text{CNN}_1, \ldots, \text{CNN}_K$ are class-specific models.

Let $\mathcal{N} = \{1, \ldots, N\}$ denote a set of classes. Denote by CNN_k the k^{th} model that includes a set of important filters denoted by $\mathcal{C}_k$ for its assigned classes. The detailed procedure to assign N classes into K models is presented in Algorithm 1. We set the initial value of K to $\lceil \frac{N}{2} \rceil$. Given a value of K, we aim at assigning N classes into K models such that the maximum number of filters across K models is minimized. Our main goal is to minimize the size and computation overhead of each model which are proportional to the number of filters, as discussed in §III-A. To achieve the goal, we first assign K classes with the highest number of filters among $\mathcal{N}$ into K models as shown in lines 7-11 in Algorithm 1. Then, we continue grouping the remaining classes in $\mathcal{N}$ into K models in multiple iterations as shown in lines 12-17 in Algorithm 1. In each iteration, we assign an additional class j to the model k that has the lowest number of filters. Specifically, among the remaining classes, the class j that shares the maximum number of the same filters with the model k is assigned to model k, i.e., $j = \arg\min_{\forall i \in \mathcal{N}} (\mathcal{F}_i \cap \mathcal{C}_k)$. The iteration stops when all remaining classes are assigned.

Finally, we perform the filter pruning denoted by *filPrune*$(\text{CNN}_0, \mathcal{C}_k)$ that removes all less important filters not in $\mathcal{C}_k$ from the CNN_0 to form the model CNN_k. The class-specific models are retrained to compensate the accuracy loss caused by the pruning. To avoid the poor accuracy caused by the unbalanced dataset, we pick the training samples of all classes assigned into the model CNN_k to form the first half of the retraining dataset. The second half with the same label consists of training samples from the remaining classes.

We define Φ_m as the maximum model size or the total size across/of K retrained models in Scenario A and B, respectively. Let $\Phi(\text{CNN}_k)$ denote the size of the model k. If the Φ_m is less than $\Phi_{\text{th}} - \delta$ where $\delta \geq 0$, K is decreased by 1 as shown in line 25 in Algorithm 1. If $\Phi_{\max} > \Phi_c$, the K is increased by 1. Then, the assignment process is repeated with the new value of K. Otherwise, the assignment process stops if $\Phi_m - \delta \leq \Phi_m \leq \Phi_{\text{th}}$. When $\Phi_m < \Phi_{\text{th}}$, increasing the number of models K by 1 may lead to the model size violation i.e., $\Phi_m > \Phi_{\text{th}}$. Thus, we use the margin $\delta \geq 1$ to avoid performing an additional assignment iteration which results in the model size violation. Moreover, the minimum and maximum numbers of models K are 1 and N, respectively.

Result Fusion: In the last step, we adopt a late fusion approach [15] that aggregates the outputs from the network's penultimate layers before the classification layers of K models to yield the classification result. Specifically, we build a multilayer perceptron (MLP) model consisting of an input layer, hidden layers, and an output layer to fuse the output information from K models. Given an input sample feeding into K models, we first concatenate the feature vectors that are the outputs of the penultimate layers of the K models. Then, the MLP takes the resulting representation vector as input to generate the final classification result.

C. Application Considerations

In this paper, we apply the proposed splitCNN framework to decompose a VGGNet [1] into multiple small-size class-specific models. As mentioned earlier, we use a different APOZ threshold to prune the filters on the different convolutional layers. From our experiments on various datasets of different sensing modalities, in VGGNet, the rear convolutional layers have higher sparsity levels of the zero activations. Thus, we use a lower APOZ threshold for pruning the less important filters in the rear convolutional layers. Moreover, the VGGNet includes a fully connected layer consisting of rectified linear units (ReLUs) before the Softmax output layer. Thus, the MLP-based fusion module takes the outputs from the fully connected layers of K class-specific VGGNet models as inputs to yield the classification result. We apply the class-specific VGGNet models obtained by the proposed splitCNN for three case studies, which are speech recognition, vibration analysis for machinery fault diagnostics, and video analytics. In particular, in the first and second case studies, we use the proposed assignment algorithm for Scenario A as shown in Algorithm 1 to split VGGNet into multiple class-specific models deployed on multiple voice-sensing IoT devices and

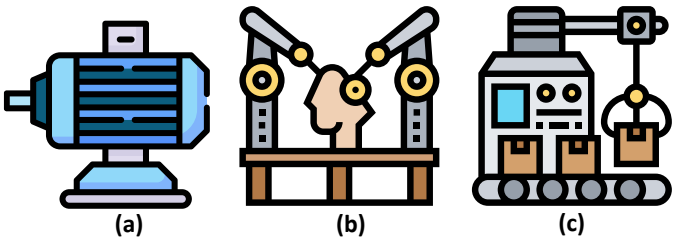


Fig. 3. Examples of vibrating objects in industrial systems. (a) Motor; (b) Assembly machine; (c) Conveyor.

vibration sensors, respectively. In the third case study, the assignment algorithm for Scenario B is used to generate multiple class-specific models deployed on a wireless camera to perform video analytics on consecutive image frames.

IV. Case Studies

A. Speech Recognition

The CNNs have been widely adopted for the speech recognition applications such as KWS [16] and ASR [17]. For instance, Google [16] developed a deep CNN model for predicting keywords (e.g., "answer call", "next song", and "pause music") in its KWS systems. The developed CNN model can achieve a relative improvement up to 44% in the false reject rate, compared with the Google's KWS approach based on a deep feed-forward fully connected neural network. The authors in [17] have designed efficient CNN-based ASR system that achieves better performance than the traditional approaches based on hidden Markov and Gaussian mixture models.

The IoT applications based on human voice interactions often require high accuracy and real-time response for good user experience. Thus, running the compute-intensive CNN-based speech recognition functions on IoT devices (e.g., smart phones and smart remote controller) is not desirable because these devices are often powered by batteries with finite capacities. For instance, due to the always-on nature, executing the CNN-based KWS function on such devices may require bulky batteries or wired power supply. A common solution is to offload the audio streams to the cloud, in which the CNN is used for speech recognition. However, this solution may suffer from the long latency and privacy concerns.

To address the above challenges, in this paper, we propose a collaborative CNN-based speech recognition systems, in which multiple IoT devices run the lightweight class-specific models obtained by splitCNN, as illustrated in Fig. 1. The devices can run the class-specific models to collaboratively perform the KWS function. Given a human voice command, each device runs the inference using its class-specific model. The outputs are fused to yield the final KWS result at a centralized node. Another potential application scenario is that the users in a meeting use their smartphones to collaborate for ASR-based functions such as automatic transcription. By enabling the execution of the small-size models on concurrent voice sensing devices, our proposed approach can perform the speech recognition with low latency.

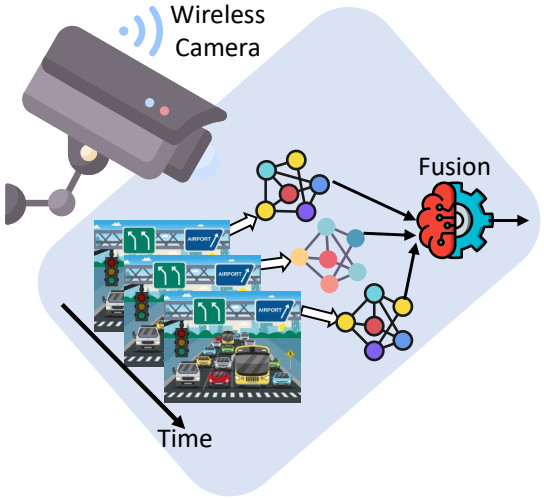


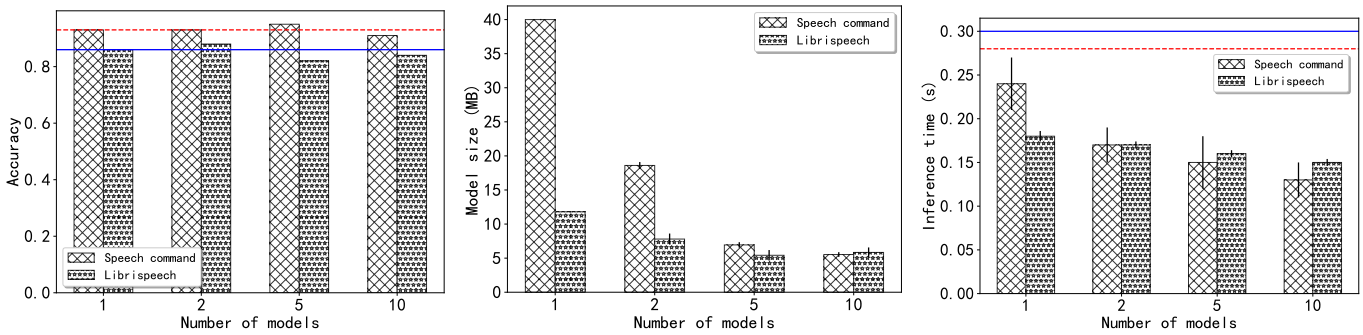
Fig. 4. splitCNN for low-power video analytics on a wireless camera.

B. Vibration Analysis for Industrial Systems

Vibration is an important sensing modality in industrial systems. In particular, the vibration signals of industrial objects (e.g., the motors, assembly machines and conveyors as illustrated in Fig. 3) reflect their internal states. The malfunction of such vibrating objects usually results in abnormal changes in the amplitude and frequency of the vibration signals [11]. Thus, the vibration analysis is an essential task in various industrial predictive maintenance applications such as diagnosing the bearing faults [18] and monitoring the machinery health [11].

CNNs have been adopted to develop various effective approaches to analyze the vibration data [18] to detect abnormal states of monitored industrial objects. For instance, the authors in [19] trained a VGG-19 model which takes the vibration signals as inputs to diagnose the faults of the rotating machines. In [20], a CNN based on LetNet-5 was developed to analyze the vibration for the industrial fault diagnosis and achieve an accuracy up to 99.79%. Such CNN-based vibration analysis approaches are often executed on a centralized resourceful node. However, collecting the high-rate vibration measurement data from the sensors to the centralized node is a challenging task especially when wireless communication is adopted. This is because the industrial spaces typically have noisy and time-varying wireless channels due to the moving parts of production lines and noises from the working machines.

In this paper, we apply our splitCNN to design a vibration sensing system where multiple sensors are deployed at different locations of the monitored object. Each sensor is equipped with an embedded computing unit to execute the lightweight class-specific CNN model obtained by splitCNN. Compared with the traditional approach, our proposed system may require more sensors for the collaborative CNN-based vibration analysis. However, our approach obviates the need of energy-intensive data transmission. Moreover, running the lightweight model to perform in-situ vibration analysis can achieve low-latency monitoring, which is important for the time critical industrial systems.



(a) Test accuracy. (b) Average, min and max of model size. (c) Average, min and max inference times.

Fig. 5. Performance of the original VGG-19 and class-specific models on speech datasets. In (a) and (c), the dotted red and solid blue lines represent results of the original VGG-19 on Google command and LibriSpeech datasets, respectively. In (b), the size of the original VGG-19 is 241MB and 267MB on Google command and LibriSpeech datasets, respectively.

Our system design is based on an assumption that the sensors can obtain the same vibration signal. However, the sensor deviation and inconsistency may pose challenges for the deployment of our system in practice. For instance, at runtime, the sensors may have different vibration readings due to calibration issues and random environment noises. Such sensing deviations across the sensors may negatively affect the accuracy of our approach. Thus, advanced signal preprocessing may be required before feeding the vibration date into the CNN models to maintain satisfactory accuracy.

C. Low-Power Video Analytics on Wireless Cameras

Today, wireless cameras have been widely deployed for various 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 at multiple locations in a monitored space, thus providing wider coverage with better view angles. The CNNs have shown outstanding performance for video analytics [21]. However, running the complex CNN-based video analytics may not be feasible on the wireless cameras with limited computation and power resources. Offloading the video data to a remote resourceful node also suffers from several issues such as high latency and communication bandwidth usage.

The conventional video analytics systems often adopt high image frame rates to minimize the miss rate in detecting the interested objects [21]. As a result, the consecutive frames have high temporal correlations, i.e., they often have similar contents. Thus, running the large CNN to process these frames for video analytics may incur a prohibitive computation overhead but not contribute to the accuracy improvement. Existing studies (e.g., [21]) have proposed various approaches to adapt the frame rate at runtime with the objective of avoiding capturing consecutive frames with the same contents. However, the design of these approaches are non-trivial. In this paper, we propose to use the small-size models obtained by the proposed splitCNN to process the consecutive frames with similar contents in the video clips.

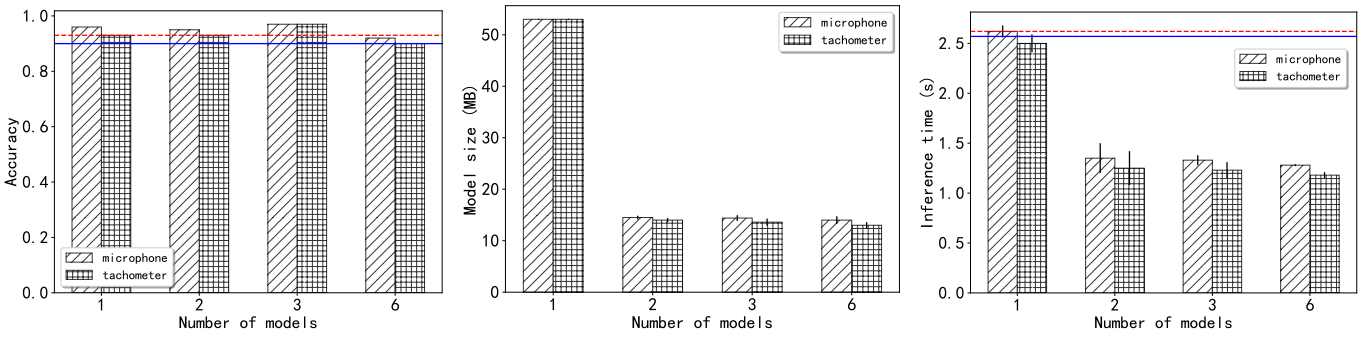
As illustrated in Fig. 4, the camera hosts all K class-specific models that are sequentially executed to process the consecutive frames in the recorded video clip. The outputs of these models on K consecutive frames are fused to generate the video analytics result (e.g., object detection). Different from the traditional approaches [21] that run the large multi-class CNN model to process all images in the video, our approach executes the small-size CNN models on the consecutive frames with ssimilar contents. As a result, the low-power and low-latency video analytics can be achieved on the wireless camera with limited computing and power resources.

V. Evaluation

In this section, we conduct experiments to evaluate the performance of the proposed splitCNN for the three case studies. In particular, we apply the proposed splitCNN to decompose a original VGG-19 model without the batch normalization into multiple class-specific models. We use TensorFlow (TF) 2.1 to implement the VGG-19, then prune the filters and retrain the class-specific models in Python 3.7. For the filter pruning, we set the APOZ threshold for the first convolutional layer to 60%. The threshold is decreased by 3% for each subsequent layer. For the fusion, we build an MLP which consists of an input layer, two hidden layers, and an Softmax output layer. The first and second hidden layers have 512 and 246 rectified linear units (ReLUs), respectively. Moreover, we use test accuracy, model size, and inference time as the evaluation metrics. Specifically, the mode size is the amount of memory required to store and run the trained model, while the inference time is the total execution time of the class-specific model and the MLP on a Raspberry Pi 4.

A. Case Study 1: Speech Recognition

We use Google Speech Command [22] and LibriSpeech [23] that are two standard datasets for the KWS and ASR, respectively. Specifically, we use Version 2 of Google Speech Command which consists of 105,000 one-second audio utterances. It contains 35 keywords (e.g., “yes”, “no”, “one”, and “two”) each of which has 1,500 samples. LibriSpeech contains about 1,000 hours of English speech corpus sampled at 16 ksps. For each audio sample, we calculate the 40-dimensional Mel-Frequency Cepstral Coefficients (MFCC) frames. Eventually, each sample in Command and LibriSpeech datasets is



(a) Test accuracy. (b) Average, min and max of model size. (c) Average, min and max inference times.

Fig. 6. Performance of the original VGG-19 and class-specific models on vibration datasets. In (a) and (c), the dotted red and solid blue lines represent results of original VGG-19 on microphone and tachometer measurements, respectively. In (b), the size of the original VGG-19 is 864MB and 841MB on microphone and tachometer measurements, respectively.

converted to $40 \times 44 \times 1$ and $40 \times 50 \times 1$ MFCC tensors, respectively. For the experiments with the above two datasets, we use all samples of ten classes.

Fig. 5 shows the accuracy, model size, and inference time of the proposed splitCNN and the original VGG-19. With our proposed splitCNN, the number of class-specific models K varies from 1 to 10. Given a value of K, the ten classes are assigned into K models using Algorithm 1. Specifically, with $K = 1$, only one 10-class model is formed by pruning all less important filters from the original VGG-19. With $K = 10$, the original model is decomposed into ten binary models, each of which is sensitive to one class. The inference time is the largest execution time across the K models plus the execution time of the MLP per voice sample.

From Fig. 5(a), the proposed splitCNN can always maintain high accuracy similar to that of the original VGG-19 on both two Command and LibriSpeech datasets. Moreover, Figs. 5(b) and 5(c) show that the model size and the inference time of splitCNN decrease with the number of models. This is because with more models, each class-specific model is assigned with fewer classes, which leads to fewer important filters to be kept. As a result, the model size and computation overhead are reduced. However, since each class-specific model is deployed on a sensor, more sensors are required when more class-specific models are generated.

B. Case Study 2: Vibration Analysis

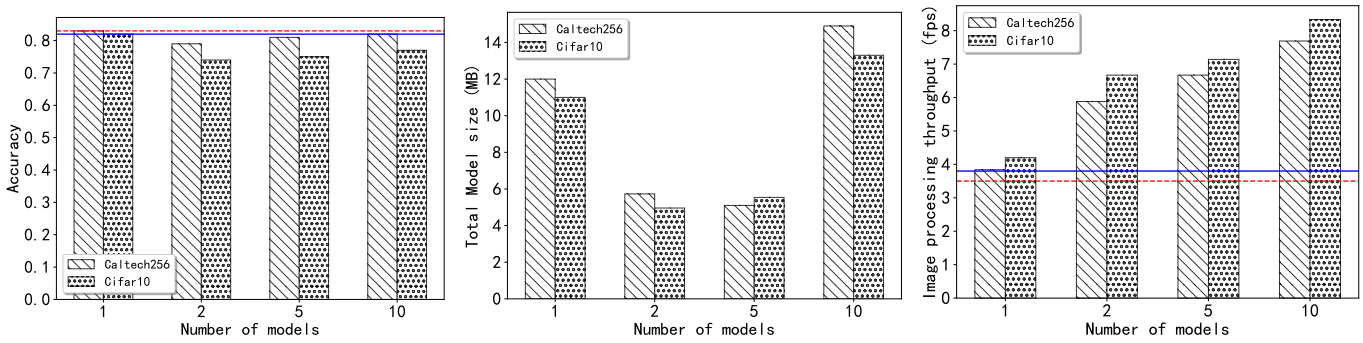
For the vibration analysis, we use a machinery fault database (MaFaulDa) [24] that consist of 1,951 samples measured by eight vibration sensors, including six accelerometers, a tachometer and a microphone attached on a machinery fault simulator. Specifically, eight sensors concurrently measure the vibration signals under six different machinery states which are normal function, inner and outer bearing faults, imbalance fault, horizontal and vertical misalignment faults, i.e., the dataset consists of six classes. For each machinery state, the measurements of a sensor contain eight traces, each of which was sampled at 50 ksps during 5s, resulting in a total of 250,000 samples. In our experiments, we use the vibration samples measured by the microphone and tachometer as two different sensing modalities to predict the machinery states. With each sensor dataset, we convert 250,000 samples into $224 \times 224 \times 1$ samples and duplicate each sample twice to form a three-channel image sample like an RGB image, which is fed to the CNN model for detecting the machinery state. Eventually, we have a total 294 samples that are divided into the training and testing datasets by following a ratio of 9:1.

Fig. 6 presents the performance of our approach and the original VGG-19 on testing vibration samples. Similar to the speech recognition in the case study 1, our proposed approach can also achieve significant reductions in both the model size and the inference time while maintaining the accuracy level of the original VGG-19 in detecting the machinery state based on the vibration signals.

C. Case Study 3: Video Analytics

Lastly, we evaluate the performance of our approach for the image classification task in the video analytics. We use Caltech256 [25] and CIFAR-10 [26] image datasets. Caltech256 consists of 30,607 images in 256 classes, each of which contains a different number of images from 80 to 827. We select a total of 3,706 images from 10 classes with the highest number of image samples for our evaluation. We use 2,964 and 742 images as the training and testing samples, respectively. CIFAR10 consists of 60,000 32×32 color images in 10 classes. We use 50,000 and 10,0000 images for training and testing datasets, respectively.

As shown Fig. 7(a), the proposed splitCNN mostly achieves the accuracy similar to that of the original VGG-19. Different from the case studies 1 and 2, in which each class-specific model is deployed on one sensor, in this case study, all models are installed on one camera. Thus, in Fig. 7(b), we present the total memory required to store all class-specific models. As shown in Fig. 7(b), the total size of all class-specific models is always less than the size of the original VGG-19 under the different settings of K. When $K < 10$, more class-specific models lead to lower total memory usage. In addition, we present the image processing throughput in frame per second (fps) in Fig. 7(c). From Fig. 7(c), splitCNN always achieves higher throughputs, compared with the original model. The reason is that with splitCNN, the camera runs a small-size class-specific model to process each image frame. As result,



(a) Accuracy. (b) Total size of class-specific models. (c) Average, min and max of inference times over testing samples.

Fig. 7. Performance of the original VGG-19 and class-specific models on image datasets. In (a) and (c), the dotted red and solid blue lines represent the results of the original VGG-19 on Caltech256 and Cirfar10 datasets, respectively. In (b), the size of the original VGG-19 is 241MB on both two datasets.

the high image processing throughput can be obtained. Moreover, the throughput of splitCNN increases with the number of class-specific models.

In our evaluation experiments, with a certain number of class-specific models, the same image frames are always fed into these models. Thus, the accuracy of splitCNN remains stable under various settings of the model number as shown in Fig. 7(a). However, in practice, more class-specific models may lead to decreased accuracy since the consecutive frames fed into the class-specific models may have less similar contents. Therefore, in the practical deployment, the number of class-specific models needs to be carefully determined to balance the trade-off between the accuracy, the memory usage and the processing throughput.

VI. Conclusion

This paper designed splitCNN, a CNN model splitting framework that allows executing the CNN-based advanced data analytics on a collection of concurrent IoT sensors. We applied a CNN pruning technique to decompose a complex CNN into multiple lightweight class-specific models that can be deployed on the resource-constrained sensors. The sensors use the class-specific models to perform in-network, low-latency data analytics on their measurement data. Extensive evaluation on three case studies and comparisons with the original CNN show the effectiveness of our proposed splitCNN approach. Specifically, our approach can achieve significant reductions in the model size and the inference time while maintaining the accuracy at the level of the original CNN model.

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] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE CVPR*, 2015, pp. 1–9.

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv:1510.00149*, 2015.

[5] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv:1506.02626*, 2015.

[6] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation."

[7] 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.

[8] 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.

[9] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *IEEE ICCV*, 2017.

[10] B. Fang and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *MobiCom*.

[11] S. W. Doebling, C. R. Farrar, M. B. Prime, and D. W. Shevitz, "Damage identification and health monitoring of structural and mechanical systems from changes in their vibration characteristics: a literature review."

[12] 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.

[13] H. Hu, R. Peng, Y. Tai, and C. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016.

[14] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv:1608.08710*, 2016.

[15] M. Seeland and P. Mäder, "Multi-view classification with convolutional neural networks," *PLoS ONE*, vol. 16, no. 1, 2021.

[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. Zhang, S. Zhang, B. Wang, and T. G. Habetler, "Deep learning algorithms for bearing fault diagnostics—a comprehensive review," *IEEE Access*, vol. 8, pp. 29 857–29 881, 2020.

[19] 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.

[20] 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, no. 7, pp. 5990–5998, 2017.

[21] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *SIGCOMM'18*.

[22] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[23] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *IEEE ICASSP*, 2015.

[24] https://bit.ly/3l3D6cj.

[25] A. P. P. Griffin, G. Holub, "The caltech 256," Techical Report, 2007.

[26] A. Krizhevsky, "Learning multiple layers of features from tiny images," Techical Report, 2009.