Towards Transmission-Friendly and Robust CNN Models over Cloud and Device

Chuntao Ding, Zhichao Lu, Felix Juefei-Xu, *Member, IEEE,* Vishnu Naresh Boddeti, *Member, IEEE,* Yidong Li, *Senior Member, IEEE,* Jiannong Cao, *Fellow, IEEE*

Abstract—Deploying deep convolutional neural network (CNN) models on ubiquitous Internet of Things (IoT) devices has attracted much attention from industry and academia since it greatly facilitates our lives by providing various rapid-response services. Due to the limited resources of IoT devices, cloud-assisted training of CNN models has become the mainstream. However, most existing related works suffer from *a large amount of model parameter transmission and weak model robustness*. To this end, this paper proposes a cloud-assisted CNN training framework with low model parameter transmission and strong model robustness. In the proposed framework, we first introduce MonoCNN, which contains only a few learnable filters, and other filters are nonlearnable. These nonlearnable filter parameters are generated according to certain rules, i.e., the filter generation function (FGF), and can be saved and reproduced by a few random seeds. Thus, the cloud server only needs to send these learnable filters and a few seeds to the IoT device. Compared to transmitting all model parameters, sending several learnable filter parameters and seeds can significantly reduce parameter transmission. Then, we investigate multiple FGFs and enable the IoT device to use the FGF to generate multiple filters and combine them into MonoCNN. Thus, MonoCNN is affected not only by the training data but also by the FGF. The rules of the FGF play a role in regularizing the MonoCNN, thereby improving its robustness. Experimental results show that compared to state-of-the-art methods, our proposed framework can reduce a large amount of model parameter transfer between the cloud server and the IoT device while improving the performance by approximately 2.2% when dealing with corrupted data. The code is available at https://github.com/evoxlos/mono-cnn-pytorch.

Index Terms—Internet of Things, cloud computing, cloud-assisted, CNNs.

1 Introduction

Background & Motivation. With the advent of the Internet of Everything era hundreds of millions of Internet of Things (IoT) devices will be connected to the network. With the excellent performance of the deep convolutional neural networks (CNNs) in computer vision [1], [2], speech [3], natural language processing [4], [5], deploying CNNs on IoT devices can provide various convenient services [6]–[11]. Limited by the insufficient resources of IoT devices, the method to successfully benefit from the excellent performance of CNNs is to seek well-resourced cloud servers to assist in training CNN models.

Fig. 1 shows the process of cloud-assisted CNN model training. The system architecture consists of two components: IoT devices and cloud servers. The deep CNN model is trained in the cloud server and then sent to the IoT device to provide users with services. When the subsequent CNN model is updated, the cloud server will periodically deliver the updated model to the IoT device. To combine ubiquitous IoT devices with high-performance CNN models and provide users with high-quality services this paper will study the cloud-assisted training of CNN models.

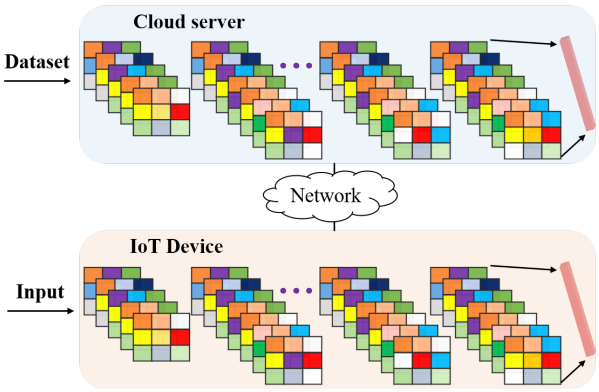


Fig. 1: System architecture for cloud-assisted training of CNN models.

Challenges. Implementing cloud-assisted training of a CNN model system is a nontrivial task that faces the following two key challenges: The first key challenge is to *reduce the model parameters* sent by the cloud server to the IoT devices. The cloud server usually assists hundreds of millions of IoT devices in deploying and updating CNN models. In addition, the number of parameters of the deep CNN model is high. During model training or subsequent model updating, frequent and large numbers of model parameter exchanges will place considerable pressure on the network bandwidth. Therefore, reducing the amount of model parameter trans-

- *Chuntao Ding and Yidong Li are with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. E-mail: {chtding, ydli}@bjtu.edu.cn.*
- *Zhichao Lu is with the School of Software Engineering, Sun Yat-sen University, Guangzhou 510006, China. E-mail: luzhichaocn@gmail.com. (Corresponding author: Zhichao Lu)*
- *F. Juefei-Xu is with Alibaba Group, USA. E-mail: juefei.xu@gmail.com.*
- *Vishnu N. Boddeti is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, 48824, USA. E-mail: vishnu@msu.edu.*
- *Jiannong Cao is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: csjcao@comp.polyu.edu.hk.*

mission is a prerequisite for the smooth progress of cloud-assisted training of CNN models in the era of the Internet of Everything.

The second key challenge arises from *enhancing the robustness* of the CNN model on the IoT device. Due to the universality of the distribution of IoT devices, the input data for the execution of tasks are prone to degradation of CNN model performance due to environmental influence or man made malicious attacks. For example, the image data obtained on a rainy or a snowy day or the image is slightly enlarged, or some pixels are removed. Ensuring the robustness of the CNN model is a practical problem. The robustness of a model in this paper refers its generalization performance against corrupted data. Therefore, ensuring the robustness of the CNN models deployed on IoT devices is the key to its deployment.

Our solutions. To address the first challenge, we propose MonoCNN. In MonoCNN, we only learn a *single filter in each layer*, referred to as the seed filter, and generate the other parameters of the layer through a seed filter and filter generation function (FGF). The parameters of FGF are randomly generated and fixed, which allows them to be reproducible with a few random seeds. Therefore, the cloud server only needs to send these seed filters and random seeds to the IoT device, and the trained MonoCNN model can be reproduced on the IoT device. Compared with sending all the model parameters, sending these seed filters and seeds can significantly reduce the number of parameters transmitted from the cloud server to the IoT device.

To address the second challenge, we propose that the parameters of the MonoCNN do not completely depend on the training data. In the MonoCNN, only the parameters of the seed filter are obtained through training, and the other parameters are obtained through FGF. This makes MonoCNN affected not only by the training data but also by the rules of the FGF. As a result, our MonoCNN naturally avoids overfitting through FGF regularization so that it has better generalization when inputting corrupted data. We also investigate five FGFs and find that the monomial function significantly outperforms the others.

In summary, our main contributions are as follows:

- To the best of our knowledge, this is the first work that seeks to reduce model parameter transmission when training CNN models in a cloud-assisted way. Our key idea is to issue only a small number of seed filters and seeds and improve model robustness by incorporating filter generation function rules.
- We perform a theoretical analysis of the MonoConv layer, showing that it can approximate the standard convolutional layer well.
- The experimental results show that the proposed framework reduces a large amount of model parameter transfer between the cloud server and the IoT device and improves the mean accuracy by approximately 2.2% when dealing with corrupted data.

The rest of the paper is organized as follows: Section 2 reviews related work. Section 3 describes the proposed framework. Section 4 presents our evaluation results. Finally, we conclude this paper in Section 5.

2 Related Work

Combining cloud servers, Internet of Things (IoT) devices, and deep neural network models to provide users with high-quality services has become mainstream. We group existing work into three categories (cloud-only, device-only, and cloud-device collaboration) based on where the neural network model training and inference are performed.

Cloud-only: The key desiderata of generating a high-performance neural network model are sufficient computing resources and sufficient training data. The configuration of the cloud server perfectly matches these desiderata, which introduced research on running neural network models in the cloud server [12]–[15]. Among them, Jiang *et al.* [12] proposed a video analysis controller based on cloud and deep neural networks. Liu *et al.*. [13] proposed a deep learning-based food recognition system that runs a deep neural network model in the cloud, and the device obtains recognition services by uploading the collected data to the cloud. To reduce the quantity of data uploaded to the cloud, they also incorporate edge computing to process data on edge servers [16]–[18]. However, cloud execution is highly dependent on network conditions. When network conditions are unstable or disconnected, cloud-based deep neural network models become degraded or unavailable.

Device-only: With the enhancement of computing and storage capabilities of IoT devices, it is possible to train neural network models directly on IoT devices, which has also led to the birth of many excellent lightweight models, such as MobileNets [19]–[21], resource-aware models [22], inference efficiency [23], [24] and others [25]. For example, Howard *et al.* [19] used depthwise separable convolution instead of standard convolution to reduce the number of parameters in the network model. Fang *et al.* [22] deployed many models on end-devices and nested these models together to provide users with multiple model choices while saving storage and switching overhead. Teerapittayanon *et al.* [24] and Fang *et al.* [23] introduced an early exit and multi-branch network to improve the efficiency of inference. The above methods explore how to modify the neural network model to adapt to the IoT device or better training or inference. Hence they are complementary to our proposed approach.

Cloud-device collaboration: Research on cloud-device collaborative training and inference has received high levels of attention with a large number of excellent approaches [26]–[33] have been proposed. For example, Zhang *et al.* [26] train the neural network model through cloud-edge collaboration and prune the deep neural network in the cloud to minimize the number of model transmission parameters while retaining the original model performance to the greatest extent. Stefanos *et al.* [27] proposed a progressive inference method for collaborative device and cloud computing and used compression [34] and quantization [35] to reduce the amount of parameter exchange between the device and the cloud. Kang *et al.* [30] divided the CNN into a head that runs on the device and a tail that runs on the cloud and decided the split point according to the load of the device and the cloud and network conditions. Akin, Li *et al.* [32] proposed a joint accuracy and latency-aware execution framework, which explores the splitting points of neural network models so that one part runs on

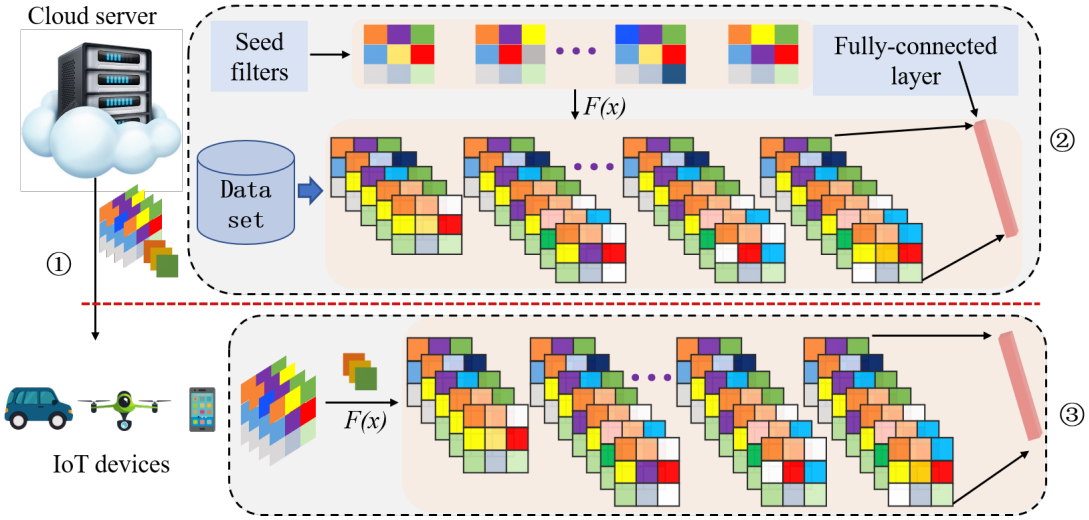


Fig. 2: Overview of the proposed framework. In the proposed framework, we first design the cloud-assisted method. Then, we design a seed filter-based CNN (i.e., MonoCNN). Each layer of MonoCNN only needs to learn the weights of one filter (i.e., seed filter), which makes the cloud server only need to send the seed filters and a small number of seeds to the IoT device. Finally, the IoT device generates the MonoCNN trained by the cloud server according to the seed filters, seeds and filter generation function.

edge devices and the other part runs in the cloud, achieving fewer parameter exchanges. The above methods have made great contributions to reducing model parameter exchange. However, some methods reduce the exchange of model parameters by finding the best splitting point. Since the optimal splitting points of different models are different, it is time-consuming and labor-intensive to search for suitable splitting points. In addition, reducing the transmission of model parameters through compression and quantization results in a loss in model performance.

In contrast, our proposed approach only requires the cloud server to send one seed filter and one seed for each layer in MonoCNN to the IoT device, which solves the overload of network transmission bandwidth caused by excessive model parameter transmission. In addition, we also address the challenge of ensuring model robustness, which is ignored by the above approaches, by using rules to regularize the generation of model parameters.

3 Design of the Proposed Approach

3.1 Overview

Fig. 2 illustrates the architecture of the proposed framework. In our framework, we first train MonoCNN in the cloud server. Instead of learning all the parameters of MonoCNN, we only learn a single filter in each layer, referred to as the seed filter. Other parameters of each layer are generated by its corresponding seed filter and filter generation function (FGF) and can be reproduced through a random seed. Therefore, the cloud server only sends a small number of seed filters and seeds to the IoT device. After obtaining seed filters and seeds, the IoT device uses them to generate multiple novel filters and combine them into the MonoCNN trained on the cloud server. We describe the proposed framework in detail as follows: ①: the cloud-assisted method in Section 3.2, ②: seed filter learning in Section 3.3, and ③: the filter generation function in Section 3.4.

3.2 Design of cloud-assisted method

Our goals are threefold i) leverage the complete resources of cloud servers, ii) limit the demand for resources on IoT devices, and iii) minimize the amount of model parameter transmission. To facilitate this goal, we propose training MonoCNN on the cloud server first and then sending the trained model to the IoT device for deployment. In general, one cloud server corresponds to millions of IoT devices. We train MonoCNN on the cloud server and then send the trained MonoCNN to IoT devices, which facilitates updating and maintaining our model on the IoT device.

3.3 Design of seed filter learning

Existing high-performance CNN models have a large number of parameters. For example, VGG19 [1] has 144 million parameters. In addition, one cloud server corresponds to millions of IoT devices. Thus a large number of model parameters still need to be transmitted to IoT devices. To this end, we start with an analysis of the CNN model parameters. As [36] shows, the standard CNN model contains many redundant parameters [2]. To reduce the number of learnable parameters in the CNN model, [36] first generates several filters and then generates some novel filters through inexpensive operations. Juefei et al. [37] decomposed a standard convolutional layer into two modules, a nonlearnable layer, and a 1×1 convolutional layer. Introducing the nonlearnable layer may represent a breakthrough in reducing the number of model parameters sent by the cloud server to the IoT devices. This is because the nonlearnable parameters are randomly initialized and can simply be saved and reproduced from a random seed. The nonlearnable parameters in this paper refer to the parameters in the CNN model that remain unchanged during training and inference and remain unchanged during model training on the cloud server.

The above analysis inspires us to specify that the parameters of only one filter (called the seed filter) in each layer of

the CNN model should be learnable while the parameters of all other filters are nonlearnable and are generated per certain rules based on the seed filter. In this paper, we refer to this CNN as MonoCNN. Formally, in any given layer, given the seed $\boldsymbol{w}_i$ for that layer, we can generate many new filters. The filters are generated via certain specified rules, e.g., a nonlinear transformation $\boldsymbol{v} = f(\boldsymbol{w}_i)$, where $f(w_i^j) = \text{sign}(w_i^j)|w_i^j|^\beta$ is a monomial that operates on each element of $\boldsymbol{w}_i$ and $\beta > 0$ is the exponent. The convolutional outputs are computed as follows (we consider 1-D signals for simplicity):

$$\boldsymbol{y} = \sum_{j=1}^{C} f(\boldsymbol{w}_i^j) * \boldsymbol{x}^j \tag{1}$$

where $\boldsymbol{x}^j$ is the j^{th} channel of the input image and $\boldsymbol{w}_i^j$ is the j^{th} channel of the i^{th} filter. During the forward pass, weights are generated from the seed filter and are then convolved with the inputs, i.e.,

$$z[i] = f(w[i]) = \text{sign}(w[i])|w[i]|^\beta \tag{2}$$

$$v[i] = \frac{z[i] - \frac{1}{n}\sum_i z[i]}{\left(\sum_i \left(z[i] - \frac{1}{n}\sum_i z[i]\right)^2\right)^{\frac{1}{2}}} \tag{3}$$

where we normalize the response maps to prevent the responses from vanishing or exploding and $\boldsymbol{v}$ is the normalized response map.

Therefore, for a layer in MonoCNN, by specifying a seed filter along with certain rules (e.g., monomial functions), we can generate or augment as many filters as needed. For example, assume that we need m filters in total for one layer, where these m filters are nonlearnable and are pointwise monomial transformations of the seed filter $\mathcal{W}_l$. The input image $\boldsymbol{x}_l$ is filtered by these filters to generate m response maps, which are then passed through a nonlinear activation gate, such as a rectified linear unit (ReLU) [38], and become m feature maps. Accordingly, the process of generating the feature maps can be expressed as,

$$\boldsymbol{y} = \sum_{i=1}^{m} g(f(\boldsymbol{w}_i) * \boldsymbol{x}) \tag{4}$$

where $g(\cdot)$ is a nonlinear activation, and $f(\boldsymbol{w}_i)$ is the monomial filter.

Compared to a standard CNN module with the same structure (with 1×1 convolutions), the number of learnable parameters is significantly smaller in the MonoCNN model. Let us assume that the numbers of input and output channels are C_{in} and C_{out}, respectively. Therefore, the size of each 3-D filter in both the CNN and the proposed MonoCNN is $C_{in} \cdot k \cdot k$, where k is the kernel size of the filter, and there are m such filters. The 1×1 convolutions act on the m filters and create the C_{out}-channel output. For the standard CNN, the number of learnable parameters is $C_{in} \cdot k \cdot k \cdot m + m \cdot C_{out}$. For the MonoCNN model, the number of learnable parameters is $C_{in} \cdot k \cdot k \cdot 1 + m \cdot C_{out}$. For simplicity, let us assume that $C_{in} = C_{out}$, which is usually the case for a deep CNN architecture. Then, we have the parameter saving ratio:

$$\tau = \frac{\#\text{P}_{\text{CNN}}}{\#\text{P}_{\text{MonoCNN}}} = \frac{C_{in} \cdot k \cdot k \cdot m + m \cdot C_{out}}{C_{in} \cdot k \cdot k \cdot 1 + m \cdot C_{out}} = \frac{k^2 m + m}{k^2 + m}$$

and when the filter kernel size is $k = 3$ and the number of convolutional filters required for each layer satisfies $m \gg 3^2$, we have a parameter saving ratio of $\tau = \frac{10m}{m+9} \approx 10$. It should be mentioned that our proposed MonoCNN does not include 1×1 convolutions, and thus $m = C_{in} = C_{out}$. Consequently, the parameter saving ratio τ of our proposed MonoCNN becomes equal to m, i.e., the number of filters per layer in the CNN model; for a high-performance CNN model, there are typically 32, 64, 256, 512, and 1024 filters per layer. Accordingly, our MonoCNN achieves parameter savings of approximately $32\times$, $64\times$, $256\times$ or more.

On the cloud server, MonoCNN contains only a few learnable parameters while other parameters of the model are randomly generated according to predefined rules and can be saved and reproduced through random seeds. Thus, after the cloud server has trained MonoCNN, the cloud server needs to send only the seed filters and the random seeds to the IoT device to reproduce the trained MonoCNN. Compared to transmitting all model parameters, sending only seed filters and random seeds can significantly reduce communication costs.

We further explore the use of a stagewise supervised training paradigm to assist in training the MonoCNN model. Fig. 3 depicts the training pipeline. Specifically, given a MonoCNN model as a student model, we use its counterpart CNN model (a standard CNN) as a teacher model. We group the network layers into multiple stages, such that feature maps of the same size (i.e., spatial resolution) belong to the same stage while reducing the feature map size by half in each subsequent stage. Let $\boldsymbol{z}_i$ denote the output feature maps of the teacher model in the i-th stage, and let $\boldsymbol{z}_i^p$ denote the output feature maps of the student (i.e., MonoCNN) model in the i-th stage. We use the ℓ_2-norms between $\boldsymbol{z}_i$ and $\boldsymbol{z}_i^p$ as additional losses to supervise the intermediate feature learning process. In addition, we leverage knowledge distillation (KD) [39], taking the output probabilities from the teacher model as soft labels. Therefore, the final loss that we backpropagate for training the MonoCNN model is defined as follows:

$$\begin{aligned}\mathcal{L}(x; \boldsymbol{W}) = &\frac{1}{N}\sum_{i=1}^{N} \left\|\boldsymbol{z}_i - \boldsymbol{z}_i^p\right\|_2^2 && \text{(MSE loss)}\\ &+ \ell_{\text{CE}}\big(y, q(x; \boldsymbol{W})\big) && \text{(hard loss)}\\ &+ \ell_{\text{CE}}\big(p(x), q(x; \boldsymbol{W})\big), && \text{(distill loss)}\end{aligned}$$

where x and y denote the inputs and outputs, respectively, provided by the dataset; $\boldsymbol{W}$ denotes the learnable parameters of the MonoCNN model (i.e., the seed filter parameters); $p(\cdot)$ and $q(\cdot)$ are the output probabilities of the teacher model and the student (i.e., MonoCNN) model, respectively; and ℓ_{CE} is the cross-entropy loss. Note that we also add ℓ_2-norms of the learnable parameters to prevent overfitting, which are removed in the above loss formulation for brevity. See Fig. 3 for a pictorial illustration.

Fig. 4 illustrates the performance and convergence rates of the standard CNN model, the MonoCNN model without KD, and the MonoCNN model on the CIFAR-10 dataset. As shown in Fig. 4a, when processing clean data, the MonoCNN model converges faster than the standard CNN model, but the performance is lower. However, when KD is used performance of MonoCNN model improves, and

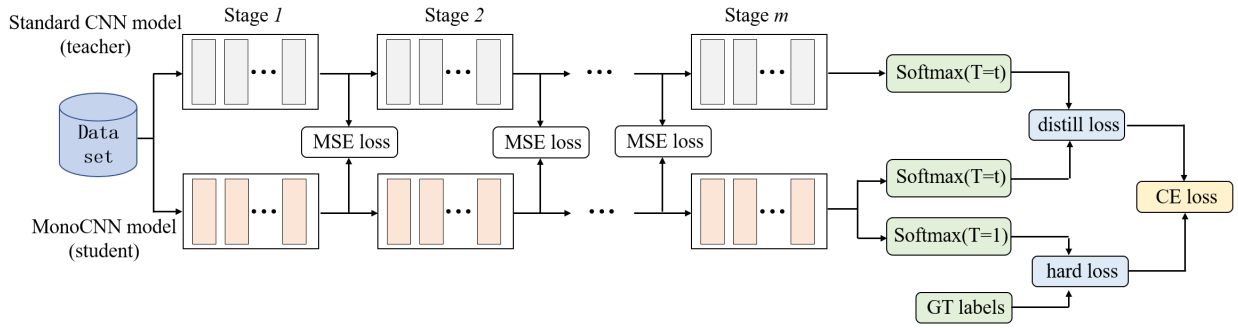
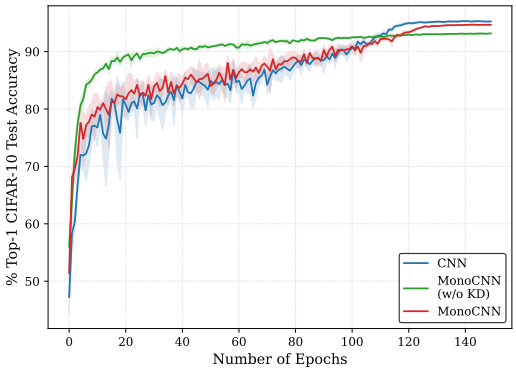
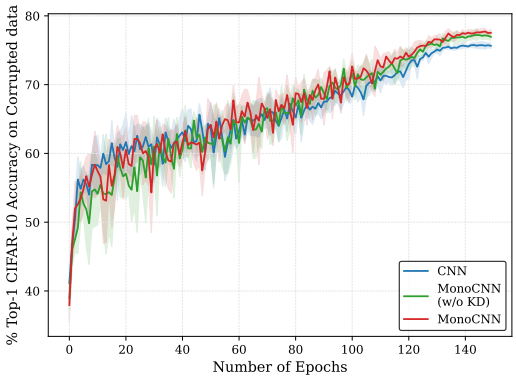


Fig. 3: Stagewise supervised training pipeline. Intermediate supervision is imposed between the feature maps of our proposed model and their counterparts.

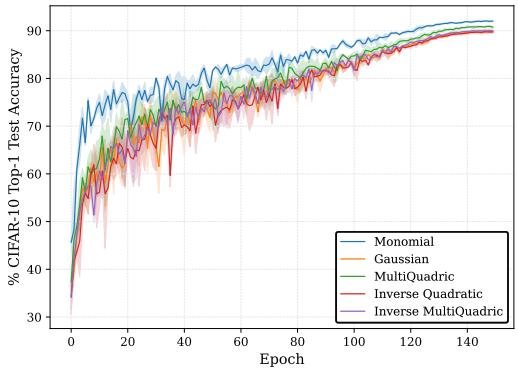


(a) CIFAR-10 data

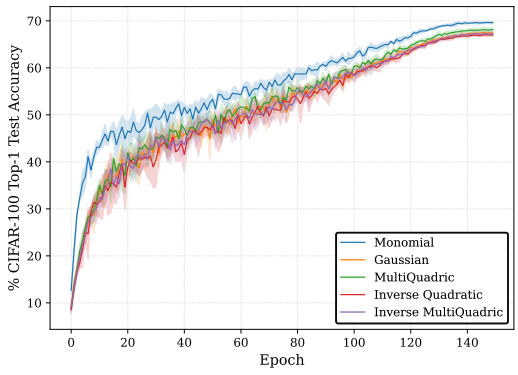


(b) CIFAR-10 corrupted data

Fig. 4: Performance and convergence rate of standard CNN, MonoCNN (without KD), and MonoCNN on the CIFAR dataset (clean vs. corrupted).



(a) CIFAR-10



(b) CIFAR-100

Fig. 5: Performance and convergence rate of five filter generation functions on CIFAR-10 and CIFAR-100 datasets.

its convergence rate decreases, possibly due to the use of a standard CNN as the teacher model. As shown in Fig. 4b, when processing corrupted data, the three methods achieve comparable convergence rates. The performance of the MonoCNN model is higher than that of the standard CNN model and slightly higher than that of the MonoCNN model without KD. The experimental results shown in Fig. 4 demonstrate the superiority of the MonoCNN model in handling corrupted data.

3.4 Filter generation function design

The combination of using seed filters and generating new filters according to certain rules makes MonoCNN comparable to or exceeds the performance of the standard CNN model in handling corrupted data. The rules for generating new filters are of great merit. The standard CNN model contains a large number of nonlinear mappings, which inspired us to use nonlinear mapping functions as FGFs. Given the existing nonlinear mapping functions and the large number of derivations included in the standard CNN model, we choose the following five functions that are easy to compute:

$$\varphi(x) = \text{sign}(x)|x|^{\beta}, \tag{5}$$

$$\varphi(x) = e^{-(\beta x)^2}, \tag{6}$$

$$\varphi(x) = \sqrt{1 + (\beta x)^2}, \tag{7}$$

$$\varphi(x) = \frac{1}{1+(\beta x)^2}, \tag{8}$$

$$\varphi(x) = \frac{1}{\sqrt{1+(\beta x)^2}}, \tag{9}$$

Eq. 5, Eq. 6, Eq. 7, Eq. 8 and Eq. 9 are monomial function, Gaussian function, multiquadric function, inverse quadratic function and inverse multiquadric function, respectively. These functions are nonlinear and easy to compute after derivation. Since a theoretical basis to prove which nonlinear mapping function is the best for generating novel filters proved elusive we empirically evaluate them across different datasets. As shown in Fig. 5, and the monomial function is significantly better than the others in terms of performance. Therefore, we use the monomial function in our FGF in this paper and call the CNN model based on the seed filter and monomial function as MonoCNN.

3.5 Discussion

3.5.1 Using MonoCNN on IoT devices

After the IoT device receives the seed filters and seeds sent by the cloud server, there are two methods for using MonoCNN. The first method is to generate the MonoCNN according to the seed filters, seeds and the FGF when the IoT is idle and store it. When the MonoCNN model needs to be used, the IoT device can page it into memory to run it in the same way as the standard CNN model. The second method is to dynamically generate the MonoCNN model. That is, when the MonoCNN needs to be used, the IoT device instantly generates the MonoCNN by paging the seed filters, seeds, and the FGF into memory. The second method, which only stores seed filters and seeds on the IoT device, can save memory usage and page-in overhead. However, the price is that there is a certain overhead in generating the MonoCNN model. Practically, since the generation process of MonoCNN has only one multiplication and addition operation, its generation overhead is small. We will test the resource overhead of generating MonoCNN on the IoT device as our future work.

3.5.2 Theoretical analysis

Here, we provide theoretical analysis on the MonoConv layer and demonstrate how it can well approximate the standard convolutional layer.

At layer l, let $\boldsymbol{x}_\pi \in \mathbb{R}^{(C\cdot k\cdot k)\times 1}$ be a vectorized single patch from the C-channel input maps at location π, where k is the kernel size of the convolutional filter. Let $\boldsymbol{w} \in \mathbb{R}^{(C\cdot k\cdot k)\times 1}$ be a vectorized single convolution filter from the convolutional filter tensor $\boldsymbol{W} \in \mathbb{R}^{C\times k\times k\times m}$, which contains a total of m generated convolutional filters at layer l. We drop the layer subscription l for brevity.

In a standard CNN, this patch $\boldsymbol{x}_\pi$ is taken as a dot product with the filter $\boldsymbol{w}$, followed by the nonlinearity (e.g., ReLU σ_{relu}), resulting in a single output feature value d_π at the corresponding location π on the feature map. Similarly, each value of the output feature map is a direct result of convolving the entire input map $\boldsymbol{x}$ with a convolutional filter $\boldsymbol{w}$. This microscopic process can be expressed as:

$$d_\pi = \sigma_{\text{relu}}(\boldsymbol{w}^\top \boldsymbol{x}_\pi) \tag{10}$$

Without loss of generality, we assume a single-seed MonoConv case for the following analysis. For a MonoConv layer, a single-seed filter $\boldsymbol{w}_s$ is expanded into a set of m convolutional filters $\boldsymbol{W} \in \mathbb{R}^{m\times k\times k\times w}$ where $\boldsymbol{w}_i = \boldsymbol{w}_s^{\circ\beta_i}$, and the exponents β_is are predefined and are not updated during training.

The corresponding output feature map value $d_\pi^{(\text{mono})}$ from a MonoConv layer is a linear combination of multiple elements from the intermediate response maps. Each slice of this response map is obtained by convolving the input map $\boldsymbol{x}$ with $\boldsymbol{W}$, followed by a nonlinearity. The corresponding output feature map value $d_\pi^{(\text{mono})}$ is thus obtained by linearly combining the m response maps with parameters $\alpha_1, \alpha_2, \ldots, \alpha_m$. This entire process can be expressed as:

$$d_\pi^{(\text{mono})} = \sigma_{\text{relu}}(\underbrace{\boldsymbol{W}\boldsymbol{x}_\pi)^\top}_{1\times m}\underbrace{\boldsymbol{\alpha}}_{m\times 1} = \boldsymbol{c}_{\text{relu}}^\top\boldsymbol{\alpha} \tag{11}$$

where $\boldsymbol{W}$ is now a 2D matrix of size $m \times k^2 w$ with m filters $\text{vec}(\boldsymbol{w}_i)$ stacked as rows, with a slight abuse of notation. $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^\top \in \mathbb{R}^{m\times 1}$. Comparing d_π and $d_\pi^{(\text{mono})}$, we consider the following two cases (i) $d_\pi = 0$: since $\boldsymbol{c}_{\text{relu}} = \sigma_{\text{relu}}(\boldsymbol{W}\boldsymbol{x}_\pi) \geq 0$, there always exists a vector $\boldsymbol{\alpha} \in \mathbb{R}^{m\times 1}$ such that $d_\pi^{(\text{mono})} = d_\pi$. However, when (ii) $d_\pi > 0$, it is obvious that the approximation does not hold when $\boldsymbol{c}_{\text{relu}} = \boldsymbol{0}$. Thus, under the assumption that $\boldsymbol{c}_{\text{relu}}$ is not an all-zero vector, the approximation $d_\pi^{(\text{mono})} \approx d_\pi$ will hold.

4 Evaluation

In this section, we first introduce our experimental setup including the datasets, baselines, and evaluation metrics studied in this work, followed by the implementation details. We then provide an empirical comparison in terms of network complexity and performance on multiple vision benchmarks.

4.1 Experimental Setup

Datasets. Five popular datasets are used to verify the effectiveness of the proposed method.

CIFAR-10/-100 [40] are two multiclass natural object datasets widely used for image classification. Both consist of 50,000 training and 10,000 test images from 10/100 classes, with each image of 32×32 pixels.

MS COCO [41] dataset comprises more than 100K images of diverse objects with annotations, including both bounding boxes and segmentation masks, from 80 categories. We take the *train2017* set for training and compare detection performance on the *val2017* set.

PASCAL VOC 2012 [42] is a comparably small-scale dataset of images with 20 foreground object categories and one category for background. Following prior works [43], we augment the original training set with the extra annotations from [44], resulting in 10, 582 images (*train_aug*) in total for training. We use this dataset for both object detection and semantic segmentation.

Cityscapes [45] is a large-scale (images are of 1024×2048 pixels) dataset for semantic understanding of urban street scenes. It is officially split into a training set of 2, 975 images, a validation set of 500 images, and a (privately hosted)

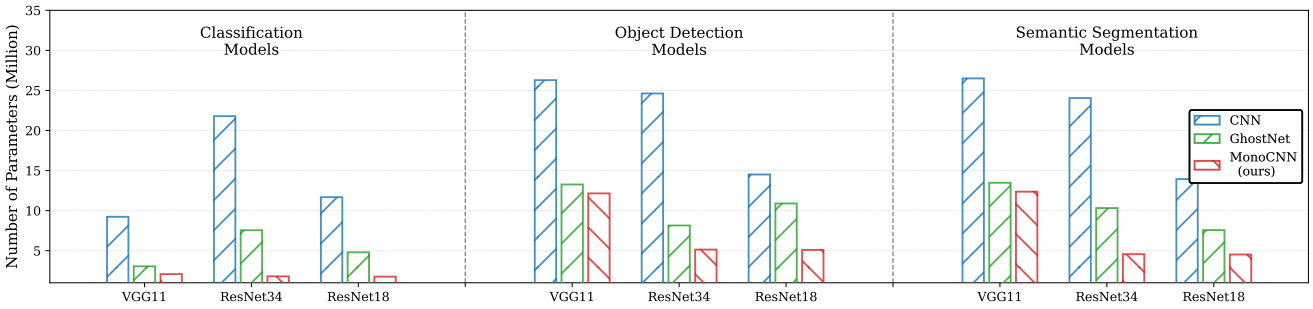


Fig. 6: Comparison of the number of parameters required to be transmitted by the regular CNN and our MonoCNN. We consider three widely used architectures and three important vision tasks. We also include GhostNet as a reference for competitor methods.

TABLE 1: Comparing MonoCNN and existing alternative methods for image classification on CIFAR-10/-100. Mean performance along with standard deviation from five runs are reported (i.e., mean$_{\pm \text{std}}$).

Method	Params (M)	Top-1 Accuracy (%)	
		CIFAR-10	CIFAR-100
CNN	9.2	92.44$_{\pm 0.19}$	71.69$_{\pm 0.10}$
LBCNN	1.2	87.67$_{\pm 0.30}$	60.57$_{\pm 0.10}$
PNN	1.2	70.61$_{\pm 0.17}$	43.56$_{\pm 0.10}$
ShiftNet	1.1	30.83$_{\pm 0.88}$	8.84$_{\pm 0.46}$
GhostNet	4.9	90.05$_{\pm 0.25}$	65.59$_{\pm 0.19}$
MonoCNN	2.4	91.45$_{\pm 0.15}$	69.17$_{\pm 0.21}$

(a) VGG 11

Method	Params (M)	Top-1 Accuracy (%)	
		CIFAR-10	CIFAR-100
CNN	11.2	95.19$_{\pm 0.11}$	77.98$_{\pm 0.33}$
LBCNN	2.8	93.05$_{\pm 0.10}$	72.72$_{\pm 0.02}$
PNN	2.8	92.45$_{\pm 0.18}$	73.21$_{\pm 0.13}$
ShiftNet	2.8	92.76$_{\pm 0.14}$	73.67$_{\pm 0.24}$
GhostNet	4.3	93.40$_{\pm 0.14}$	72.77$_{\pm 0.58}$
MonoCNN	2.8	94.02$_{\pm 0.06}$	74.22$_{\pm 0.13}$

(b) ResNet18

Method	Params (M)	Top-1 Accuracy (%)	
		CIFAR-10	CIFAR-100
CNN	21.3	95.57$_{\pm 0.08}$	78.73$_{\pm 0.42}$
LBCNN	3.9	93.54$_{\pm 0.16}$	73.81$_{\pm 0.21}$
PNN	3.9	92.31$_{\pm 0.20}$	73.33$_{\pm 0.14}$
ShiftNet	3.9	92.84$_{\pm 0.20}$	73.87$_{\pm 0.25}$
GhostNet	7.1	93.58$_{\pm 0.24}$	73.16$_{\pm 0.63}$
MonoCNN	4.0	94.24$_{\pm 0.12}$	75.63$_{\pm 0.52}$

(c) ResNet34

testing set of 1, 525 images. We use 19 from the provided 30 classes for semantic segmentation.

Baselines. To verify the effectiveness of the proposed method, we consider the following baselines:

LBCNN [37]: The local binary convolutional neural network (LBCNN) uses sparse local binary filter parameters (randomly initialized and kept fixed) followed by a learned 1×1 convolution to replace regular 3×3 convolution layers.

PNN [46]: The perturbative neural network (PNN) injects randomly generated additive noise to the input features combined through a learned 1×1 convolution to replace regular 3×3 convolution layers.

ShiftNet [47]: ShiftNet applies a sparse spatial shift (e.g., one pixel left) to create diverse viewpoints of features, replacing the regular 3×3 convolutions.

GhostNet [36]: GhostNet partially substitutes computationally expensive operations (e.g., regular 3×3 convolutions) with cheap operations (e.g., 1×1 or grouped 3×3 convolutions).

To ensure a fair and comprehensive comparison, we implement all the above baseline methods within three well-studied underlining architectures, including VGG11 [1], ResNet18 [2], and ResNet34 [2].

Evaluation Metrics. We use top-1 accuracy to compare performance for image classification. We use the mean average precision (AP), computed for a recall value over 0 to 1, for object detection. For semantic segmentation, we adopt mean intersection-over-union (mIoU), which computes the IoU for each semantic class averaged over classes. It is worth noting that we only consider the number of parameters that must be learned, as these are the parameters that must be sent from the cloud server to IoT devices.

Implementation Details. We implement our method in PyTorch 1.7 with CUDA 10.1, and all experiments are performed on 2080TI GPUs. Following the suggestions from the original papers, we set the sparsity to 0.9 for LBCNN [37] and the noise level to 0.01 for PNN [46]; we use the 1×1 convolution as the cheap operation for GhostNet [36] and set the ratio to 4.

4.2 Experimental Results

In this section, we first present a comparison of network complexity, followed by a performance comparison for image classification, object detection, and semantic segmentation on clean data. Finally, we compare robustness on limited training data, corrupted data, and different style data.

4.2.1 *Amount of Model Parameter Transmission*

Our proposed MonoCNN minimizes the number of model parameters sent by the cloud server to IoT devices. As shown in Fig. 6, we consider three widely used architectures (VGG11 and ResNet18/34) and compare the learnable parameters of MonoCNN with those of regular CNNs for image classification, object detection, and semantic segmentation. Since all filter parameters in the standard CNN model need to be learned, the cloud server needs to send all the filter parameters of the standard CNN model to the IoT device, resulting in a large amount of model parameter transmission. In contrast, in our proposed MonoCNN, only a single-seed filter needs to be learned in each layer, and the rest of the filters are generated by the filter generation function. The hyperparameters of the filter generation function (e.g., monomial exponent) are randomly initialized and remain fixed so that these nonlearnable hyperparameters can be saved and reproduced by the random number generator seed. Therefore, cloud-assisted training of MonoCNN only

requires the cloud server to send a few seed filters and the random number generator seeds to recover the MonoCNN model on the IoT device.

Additionally, as shown in Fig. 6, GhostNet also has fewer model parameters than the standard CNN model because GhostNet uses cheap operations to augment filters. However, our proposed MonoCNN needs to send fewer model parameters, and in subsequent experiments, our proposed MonoCNN outperforms GhostNet in almost all tasks. It is worth mentioning that other types of parameter reduction techniques (e.g., pruning, quantization [34], and neural architecture search [48], [49]) can be applied on top of our method for further compression of model parameters.

4.2.2 Results on Standard Benchmarks

In this section, we evaluate the effectiveness of our MonoCNN on standard benchmark datasets for image classification, object detection, and semantic segmentation tasks.

Image Classification. For training on the CIFAR-10/-100 datasets, we use the SGD optimizer with an initial learning rate of 0.025, which is annealed to zero following the cosine schedule. We use standard data augmentations: we pad images with four pixels on each side and randomly crop a 32×32 region, from which random horizontal flipping is also applied. Given the stochastic nature of the CIFAR datasets (as the results are subject to high variance even with exactly the same setup), we repeat the training five times with different initial random seeds and report the mean performance along with the standard deviation.

Table 1 depicts the results. In general, we observe that our MonoCNN consistently outperforms other peer methods on both CIFAR-10 and CIFAR-100 while requiring a similar or fewer number of parameters to be learned. Additionally, our MonoCNN provides substantial savings in the parameters while achieving similar accuracy performance when compared to regular CNNs. In particular, the proposed MonoCNN is **3.58% more accurate** on CIFAR-100 and **2$\times$ more compact** than GhostNet [36] when paired with the VGG11 architecture.

Objection Detection. To evaluate the effectiveness of our model for object detection, we implement all compared methods using ResNet18 as the underlining backbone architecture and FPN [50] as the detection head. For training on both MS COCO and PASCAL VOC 2012, we use the SGD optimizer with an initial learning rate of 0.02 and a batch size of eight over four GPU cards. Following the common practice, we adopt the $1\times$ (i.e., 12 or 36 epochs) schedule to train our detection models and decay the learning rate at the 8th and 11th epochs by a factor of 10. We resize the training images to the shorter side of 800 pixels with the longer side to be within 1333 pixels for MS COCO. We resize the training images to 1000×600 for PASCAL VOC 2012.

Table 2 and Table 3 depict the results. Similar to the previous case of image classification, the proposed MonoCNN consistently outperforms other peer methods for object detection. In particular, MonoCNN achieves **6.2 and 8.0 higher AP points** than LBCNN [37] while using a similar number of parameters. In addition, we also provide a qualitative visualization between MonoCNN and the compared methods in Fig. 7. Evidently, MonoCNN (right-most column in Fig. 7) is not only more accurate in detecting smaller objects (see the first and fourth row in Fig. 7) but also more precise in avoiding duplicate detection boxes (see second and third row in Fig. 7) than peer methods (Columns 2-4 in Fig. 7).

TABLE 2: Comparing MonoCNN and existing alternative methods for object detection on MS COCO.

Method	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
CNN	33.1	52.6	35.5	18.9	35.4	43.1
LBCNN	25.6	43.3	26.3	13.4	27.2	34.2
GhostNet	30.4	49.3	32.1	16.8	32.5	40.8
MonoCNN	31.8	51.3	34.1	17.3	33.9	41.9

TABLE 3: Comparing MonoCNN and existing alternative methods for object detection on PASCAL VOC 2012.

Method	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
CNN	47.4	79.8	50.6	19.1	33.9	52.1
LBCNN	37.7	69.5	36.1	18.0	25.1	41.8
GhostNet	44.3	76.7	45.9	17.2	31.1	48.8
MonoCNN	45.7	78.1	47.7	16.5	30.8	50.8

Semantic Segmentation. We follow the same setup as in the previous case of object detection. We also implement all compared methods using ResNet18 as the underlining backbone architecture and FPN as the segmentation head. For training on Cityscapes and PASCAL VOC 2012, we use the SGD optimizer with a momentum of 0.9 and weight decay of 5e-4. The batch size is set to 24 over two 2080TI GPUs. Following the common practice, we adopt the "poly" learning rate policy (i.e., $0.01 \times (1 - \frac{iter}{maxIter})^{0.9}$) from 0.01 to zero in 60K iterations. Data augmentation includes color jittering, random horizontal flipping, random cropping and random resizing. In addition, we scale training images with a factor randomly sampled from [0.125, 1.5] and crop them to 1024×512 for Cityscapes.

Table 4 and Table 5 break down the classwise segmentation mIoU for PASCAL VOC 2012 and Cityscapes, respectively. Evidently, we observe that our MonoCNN significantly outperforms peer competitors on both datasets. In particular, MonoCNN achieves **better mIoU with 3$\times$ fewer parameters** than the regular CNN model on PASCAL VOC 2012; MonoCNN achieves **3.6 and 5.2 points higher mIoU** than LBCNN [37] on the two datasets, respectively. A qualitative comparison is also provided in Fig. 8. Visually, we observe that MonoCNN leads to a more fain-grained segmentation on small objects (see boxed regions in 8a).

Discussion. As shown by experimental results on image classification, object detection, and semantic segmentation tasks, the proposed MonoCNN consistently outperforms a wide range of existing alternatives with similar or fewer parameters. In addition, the proposed MonoCNN can significantly decrease the number of parameters compared to standard CNN models, but with slight performance degradation. The main reason is that it is difficult for MonoCNN with a small number of learnable parameters to process the test images that are highly correlated with the training images through the training images. However, in real scenarios, in the data collected by IoT devices, the correlation between training images and test images is much smaller

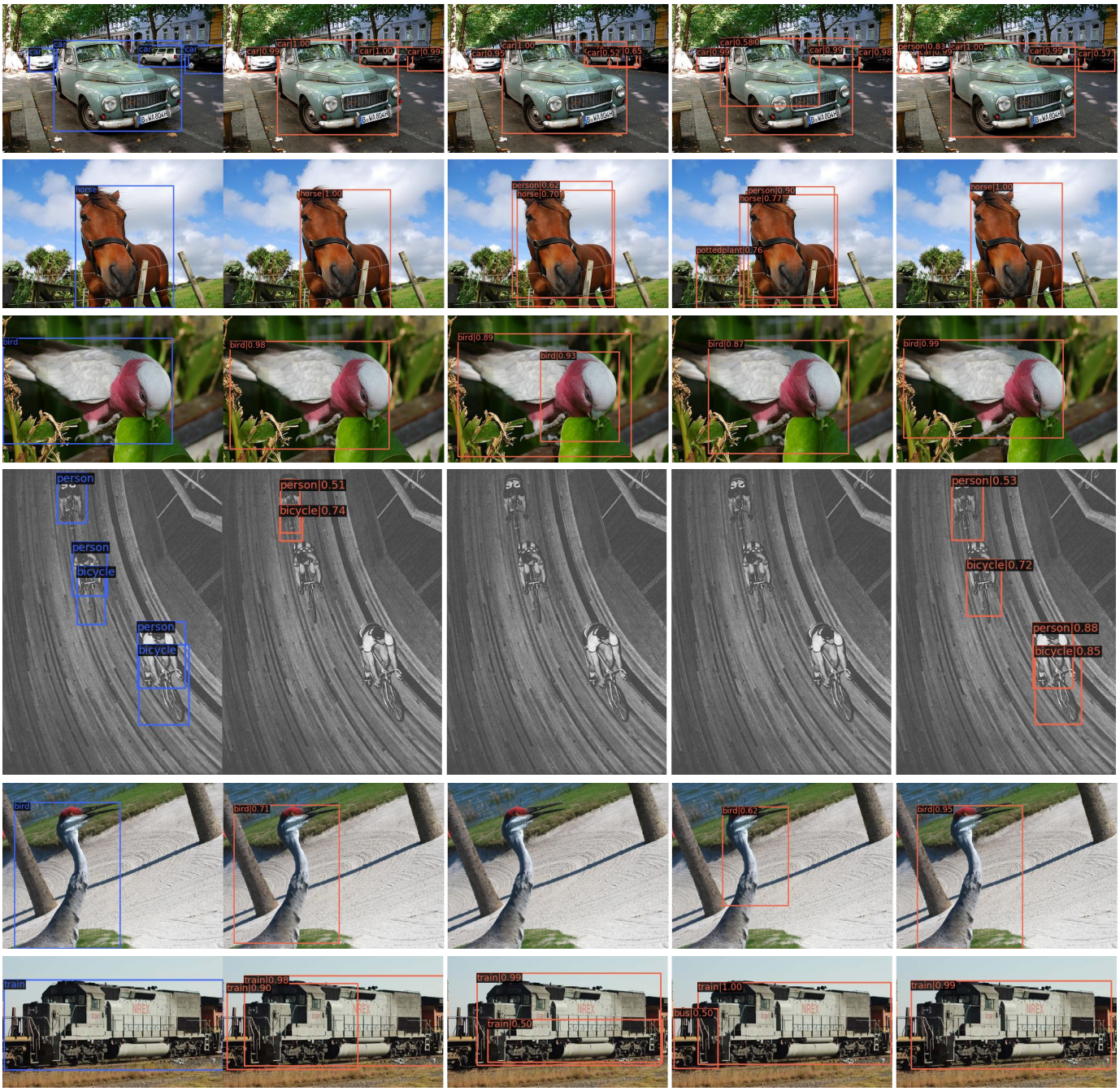


Fig. 7: Qualitative comparison on MS COCO object detection. From left to right, we show the example predictions from ground truth, regular CNN, LBCNN, GhostNet, and our MonoCNN. The predicted labels with confidence scores are annotated at the top-left corners of the detection boxes.

than that of the training image set and test image set divided by the standard dataset. For MonoCNN, its parameters are also affected by the filter generation function and are not completely dependent on the training data, thus MonoCNN is expected to achieve high performance in processing this type of image.

4.2.3 *Results on Robustness*

In this section, we use CIFAR-10 classification to evaluate the performance of the proposed MonoCNN for robustness on limited data and on data with commonly observable corruptions.

Limited training data. Insufficient training data are a conventional difficulty for deep neural network models but often arise in practical applications. Considering the lower model complexity (i.e., fewer learnable parameters), we hypothesize that MonoCNN may be less prone to overfitting to the limited training data. To verify this hypothesis, we perform an empirical experiment on (randomly selected) subsets of the CIFAR-10 training set while keeping the testing set intact. Fig. 9 depicts the results. Compared to fully learned convolutions (standard CNNs), MonoCNN exhibits noticeably better generalization performance under limited training data.

(a) Cityscapes (b) PASCAL VOC 2012

Fig. 8: Qualitative comparison on semantic segmentation. For (a) Cityscapes, we visualize the ground truth, LBCNN, GhostNet, and our MonoCNN from top to bottom. For (b) PASCAL VOC 2012, we visualize input images, ground truth, and our MonoCNN from left to right. Zoom in for details.

TABLE 4: Comparing MonoCNN and existing alternative methods for semantic segmentation on PASCAL VOC 2012.

Method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mIoU
CNN	79.6	37.4	71.4	51.3	54.2	81.5	76.5	77.6	28.1	59.9	38.3	67.8	66.6	71.7	78.4	39.2	66.4	34.4	69.7	60.4	65.2
LBCNN	71.8	35.0	58.8	47.0	47.1	76.4	73.0	72.4	21.4	46.9	37.8	59.5	51.4	63.8	71.4	31.5	64.0	30.5	65.4	52.3	61.9
GhostNet	77.4	36.2	69.3	48.8	56.4	78.4	74.7	75.8	27.1	59.1	40.8	66.0	62.6	68.5	76.0	36.3	66.5	31.3	68.4	58.8	63.5
MonoCNN	83.9	37.8	78.2	53.3	58.8	89.5	77.9	82.8	31.2	58.9	38.3	71.6	71.8	75.0	77.6	48.2	73.9	35.2	77.4	63.7	65.5

TABLE 5: Comparing MonoCNN and existing alternative methods for semantic segmentation on Cityscapes.

Method	road	sidewalk	building	wall	fence	pole	light	sign	vegetation	terrain	sky	person	rider	truck	bus	caravan	trailer	train	motorcycle	mIoU
CNN	97.3	78.8	89.9	50.1	47.4	47.4	55.5	66.8	89.9	58.6	93.0	72.7	50.3	92.4	63.3	74.0	53.4	51.0	67.9	68.4
LBCNN	96.9	76.4	88.2	47.8	42.3	40.6	44.2	59.9	88.6	56.3	92.1	66.8	42.2	90.6	56.2	65.2	36.1	41.6	63.6	62.9
GhostNet	97.3	79.4	89.2	47.9	46.5	45.7	51.7	64.7	89.6	59.9	92.6	70.7	46.6	91.5	60.0	75.8	65.7	41.4	66.2	67.5
MonoCNN	97.4	79.8	89.7	49.0	48.7	46.0	55.0	65.7	90.0	61.0	93.0	71.3	49.1	92.1	66.4	73.0	58.9	39.4	66.5	68.1

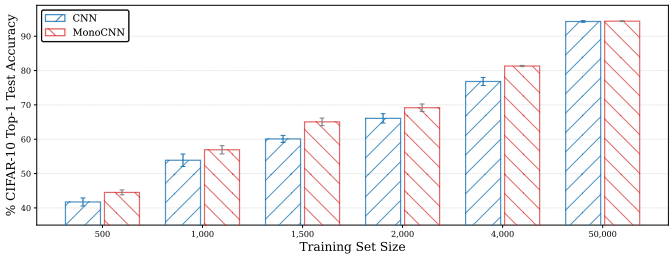


Fig. 9: Robustness to limited training data.

TABLE 6: Details of the corruption types evaluated.

Group	Corruption Types
Noise	Gaussian, Impulse, Shot, Speckle
Blur	Defocus, Glass, Motion, Zoom, Gaussian
Weather	Brightness, Fog, Frost, Snow, Spatter
Digital	Contrast, Elastic, JPEG compression, Pixelate, Saturate

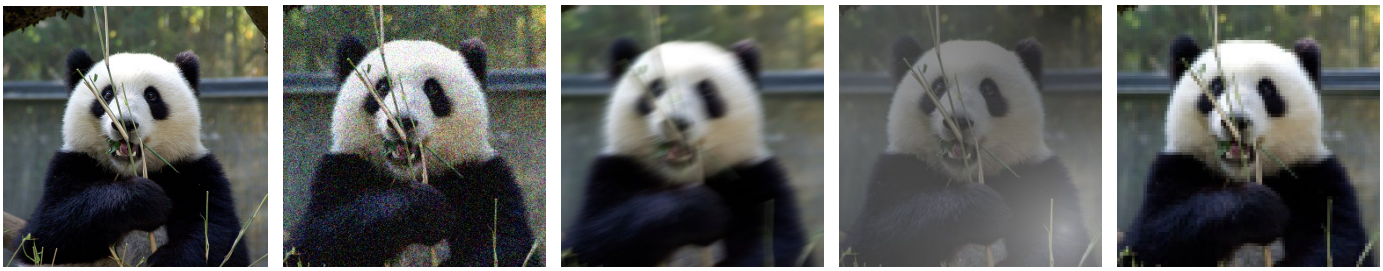
Corrupted data. The vulnerability to a small perturbation in inputs adversely affects the deployment of deep learning vision systems in many IoT applications that are sensitive to safety and user privacy. To quantitatively measure the robustness of the proposed MonoCNN, we consider the CIFAR-10-C dataset proposed by Hendrycks and Dietterich [51], who applied common observable corruption to the original (i.e., clean) test images of CIFAR-10. There are 19 different types of corruption from four main categories. See Table 6 for details and Fig. 10 for visualization.

Based on the empirical findings summarized in Table 7, we observe that our MonoCNN performs significantly better than other peer models under a similar number of parameters. In addition, MonoCNN also performs noticeably better than the regular CNN model.

TABLE 7: Robustness to commonly observable corruptions. We perform five runs and report mean performance along with standard deviation (mean$_{\pm\text{std}}$).

Method	Noise	Blur	Weather	Digital	mean
CNN	57.05$_{\pm 7.55}$	74.90$_{\pm 10.5}$	86.27$_{\pm 5.41}$	82.32$_{\pm 6.34}$	75.13$_{\pm 12.9}$
LBCNN	54.88$_{\pm 6.87}$	67.61$_{\pm 12.0}$	81.79$_{\pm 7.49}$	78.41$_{\pm 7.62}$	70.67$_{\pm 12.1}$
PNN	50.13$_{\pm 6.67}$	63.27$_{\pm 9.12}$	79.30$_{\pm 7.56}$	76.09$_{\pm 7.57}$	67.20$_{\pm 11.5}$
GhostNet	58.48$_{\pm 5.93}$	67.53$_{\pm 9.95}$	81.57$_{\pm 7.35}$	78.10$_{\pm 7.61}$	71.42$_{\pm 9.08}$
MonoCNN	64.41$_{\pm 5.42}$	77.41$_{\pm 9.41}$	85.31$_{\pm 5.01}$	82.22$_{\pm 6.22}$	**77.34**$_{\pm 7.98}$

Data under different styles. In addition to data under degraded quality, another important angle for measuring robustness is the generalization performance on data under



(a) Original (b) Gaussian noise (c) Motion blur (d) Fog (e) Pixelate

Fig. 10: Visualization examples of commonly observable corruptions shown in Table 6.

different styles, i.e., data with the same context but represented differently. We consider the Icons-50 dataset [51], which consists of 10K images from 50 classes of icons (e.g., airplane, symbols, activities, etc.) collected by various technology companies (e.g., Apple, Facebook, Google, etc.). We hold off data from one company while training on data from other companies to quantify robustness under different styles. See Fig. 11 for a visualization.

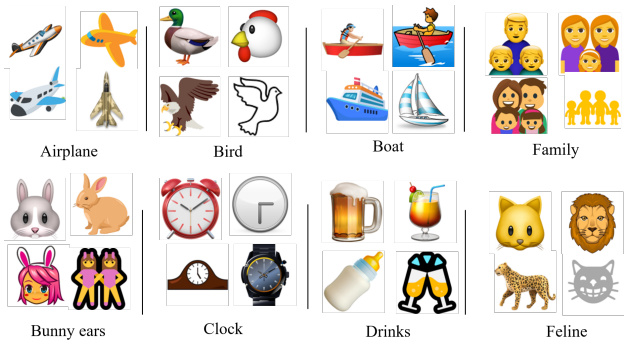


Fig. 11: Visualization examples of the Icons-50 dataset. For each class, we show images collected from Apple, Facebook, Google, and Samsung from top-left to bottom-right.

As shown in Table 8, the mean accuracy of our proposed MonoCNN outperforms the mean accuracy of other models. For example, our proposed MonoCNN achieves a mean accuracy improvement of 1.15% compared to the regular CNN. The main reason is that the parameters of MonoCNN are affected by both the filter generation function and the training data, which makes MonoCNN promising for achieving better performance than regular CNNs when dealing with test data whose style is inconsistent with the training data.

TABLE 8: Robustness to different styles. We perform five runs and report mean performance along standard deviation (mean$_{\pm \text{std}}$).

Method	Apple	Facebook	Google	Samsung	Mean
CNN	91.74$_{\pm 0.65}$	86.56$_{\pm 0.25}$	82.63$_{\pm 0.88}$	81.30$_{\pm 1.11}$	85.56$_{\pm 4.69}$
LBCNN	92.73$_{\pm 0.67}$	87.63$_{\pm 1.33}$	83.42$_{\pm 0.40}$	79.09$_{\pm 0.62}$	85.72$_{\pm 5.83}$
PNN	92.49$_{\pm 0.48}$	82.43$_{\pm 1.37}$	82.24$_{\pm 1.11}$	82.19$_{\pm 1.67}$	84.84$_{\pm 5.10}$
GhostNet	92.95$_{\pm 0.80}$	85.26$_{\pm 2.20}$	80.85$_{\pm 0.58}$	76.64$_{\pm 0.71}$	83.93$_{\pm 6.97}$
MonoCNN	93.52$_{\pm 0.59}$	86.48$_{\pm 0.66}$	82.42$_{\pm 1.61}$	84.40$_{\pm 0.87}$	**86.71**$_{\pm 4.84}$

In Table 7 and Table 8, we observe that when there exists sufficient training data and the test data are within the same underlining distribution as the training data, all efficiency-oriented methods (i.e., LBCNN, PNN, GhostNet, and MonoCNN) exhibit a lower performance due to lower model capacity from limited parameters. However, the proposed FGF mechanism provides an inductive bias to the training of MonoCNN, which prevents overfitting to the training data, in turn, leading to a better generalization performance under limited training data and on out-of-distribution test data (i.e., corrupted data or data under different styles).

4.3 Monomial Function Hyperparameter Study

As demonstrated in the previous sections, we empirically observe that the monomial transformation is better suited for the filter generation function. In this section, we perform parameter sensitivity analysis on the hyperparameters of the monomial transformation.

Effect of polynomial terms. Instead of a monomial, one may relax the constraint on the number of terms to include the more general case of polynomial transformation. Accordingly, we allow the number of terms to grow from one (i.e., monomial) to many terms and evaluate the performance of corresponding models on CIFAR-10 classification. We repeat each setup five times and present the results in Fig. 12. We observe that monomial transformation (i.e., number of terms equal to one) is better suited for filter generation function as opposed to polynomial transformation with many terms.

Effect of monomial exponent. Recall that we adopt the pointwise polynomial transformation as the filter generation function based on our empirical experiments. The monomial filter generation function randomly samples a (continuous-valued) exponent β from $[a, b]$, where a and b are the lower and upper bounds on β. To understand the effect of β, we set the number of channels to 64 and the number of layers to 20 for our MonoCNN and vary the lower and upper bounds of β. Fig. 13 depicts the results. In general, having a diverse set of exponents β (i.e.,, a larger range of β bounds) leads to better performance of MonoCNN. Empirically, we identify that setting the lower bound a to 1 and the upper bound b to 7 yields the best performance.

5 Conclusions and Future Work

In this paper, we make the following two contributions. First, we propose cloud-assisted training of a CNN model

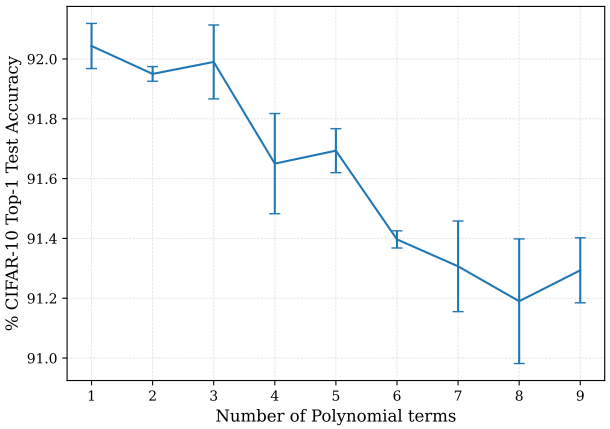


Fig. 12: Impact of the number of terms used in polynomial transformation, where monomial transformation corresponds to the number of terms equal to one.

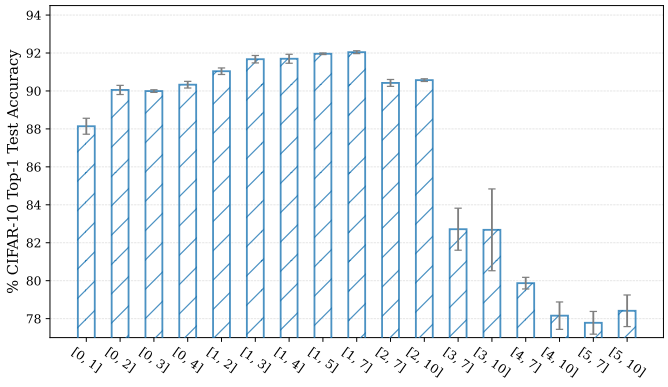


Fig. 13: Impact of polynomial exponent range. The monomial exponent β is uniformly sampled from $[a, b]$, where a and b are the lower and upper bounds.

framework for IoT devices by considering model parameter transmission and model robustness. Second, we propose a novel CNN architecture (*i.e.*, MonoCNN) that reduces the number of model parameters sent by the cloud server to IoT devices by specifying only one filter that needs to be learned in each layer of MonoCNN and improves the robustness of the model by regularizing the model parameters using the filter generation function. Experimental results show that the proposed approach achieves better performance in dealing with corrupted data and minimizes model parameter transmission.

In addition, Gill et al. [52] comprehensively combed the emerging trends and future directions of AI for next-generation computing, which motivates our future work to start from the following points:

- We will deploy MonoCNN on the IoT device (such as the Raspberry Pi 4B) and test the resources and time it takes to generate MonoCNN based on learnable parameters, seeds, and filter generation function.
- Since the available resources of the IoT device are dynamically changing, we need to deploy multiple MonoCNN variants with different capacities. However, this faces two challenges: (i) how to divide multiple MonoCNN variants with different capacities and how to train these MonoCNN variants; (ii) how to reduce the storage resources occupied by deploying multiple MonoCNN variants.
- IoT devices usually run multiple applications simultaneously. However, resources are limited. When the IoT device cannot provide sufficient resources for each application at the same time, how to reasonably allocate resources for each application poses a challenge.
- Training the high-performance MonoCNN requires a large quantity of labeled data; however, unlabeled data are common in real scenarios, and how to train MonoCNN with the help of unlabeled data is a practical challenge.
- To avoid leakage of user-sensitive private data, training MonoCNN on the IoT device is a research direction; however, how to speed up the training of MonoCNN is a challenge.

6 Acknowledgements

This work was supported by the Fundamental Research Funds for the Central Universities (2021RC272), the National Natural Science Foundation of China (62106097), the China Postdoctoral Science Foundation (2021M691424, 2021M700364), the Research Grants Council of Hong Kong through the Theme-based Research Scheme (T-41-603/20R), and the Research Grants Council of Hong Kong through the General Research Fund (PolyU 15217919).

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of 3rd International Conference on Learning Representations*, 2015, pp. 1–14.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[3] A. Bohlender, A. Spriet, W. Tirry, and N. Madhu, "Exploiting temporal context in cnn based multisource doa estimation," *IEEE/ACM Transactions on Audio, Speech and Language Processing*, vol. 29, pp. 1594–1608, 2021.

[4] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics*, 2014, pp. 655–665.

[5] S. Minaee, N. Kalchbrenner, E. Cambria, J. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning-based text classification: A comprehensive review," *ACM Computing Surveys*, vol. 54, no. 3, pp. 1–40, 2021.

[6] H. xuan Hu, Z. Jiang, Y. Zhao, Y. Zhang, H. Wang, and W. Wang, "Network representation learning-enhanced multisource information fusion model for poi recommendation in smart city," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9539–9548, 2021.

[7] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, "Collaborate edge and cloud computing with distributed deep learning for smart city internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8099–8110, 2020.

[8] Z. Lu, S. Rallapalli, K. Chan, S. Pu, and T. L. Porta, "Augur:modeling the resource requirements of convnets on mobile devices," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 352–365, 2021.

[9] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2021.

[10] X. Liang, Y. Zhang, G. Wang, and S. Xu, "A deep learning model for transportation mode detection based on smartphone sensing data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 12, pp. 5223–5235, 2020.

[11] T. Qian, C. Shao, X. Wang, and M. Shahidehpour, "Deep reinforcement learning for ev charging navigation by coordinating smart grid and intelligent transportation system," *IEEE Transactions on Smart Grid*, vol. 11, no. 2, pp. 1714–1723, 2020.

[12] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 253–266.

[13] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, M. Yunsheng, S. Chen, and P. Hou, "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 249–261, 2018.

[14] E. Georganas, S. Avancha, K. Banerjee, D. D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, "Anatomy of high-performance deep learning convolutions on simd architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2018, pp. 1–12.

[15] K. M. Hazelwood, S. Bird, D. M. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, 2018, pp. 620–629.

[16] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

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

[18] T. Liu, Y. Zhang, Y. Zhu, W. Tong, and Y. Yang, "Online computation offloading and resource scheduling in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6649–6664, 2021.

[19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," in *CoRR abs/1704.04861*, 2017, pp. 2464–2469.

[20] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[21] A. Howard, R. Pang, H. Adam, Q. V. Le, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, and Y. Zhu, "Searching for mobilenetv3," in *Proceedings of IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[22] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of 16th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 115–127.

[23] B. Fang, X. Zeng, F. Zhang, H. Xu, and M. Zhang, "Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision," in *Proceedings of 5th IEEE/ACM Symposium on Edge Computing*, 2020, pp. 84–95.

[24] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2464–2469.

[25] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 68–81.

[26] X. Zhang, M. Qiao, L. Liu, Y. Xu, and W. Shi, "Collaborative cloud-edge computation for personalized driving behavior modeling," in *Proceedings of 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 209–221.

[27] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: Synergistic progressive inference of neural networks over device and cloud," in *Proceedings of 16th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–15.

[28] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proceedings of 37th IEEE International Conference on Distributed Computing Systems*, 2017, pp. 328–339.

[29] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and efficient object detection in edge computing: Challenges and solutions," *IEEE Network*, vol. 32, no. 6, pp. 137–143, 2018.

[30] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 615–629.

[31] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *Proceedings of IEEE Conference on Computer Communications*, 2019, pp. 1423–1431.

[32] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *Proceedings of IEEE 24th International Conference on Parallel and Distributed Systems*, 2018, pp. 671–678.

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

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proceedings of 4th International Conference on Learning Representations*, 2016, pp. 1–14.

[35] I. Hubara, M. Courbariaux, D. Soudry, R. EI-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *Journal of Machine Learning Research*, vol. 18, pp. 1–30, 2017.

[36] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1577–1586.

[37] F. Juefei-Xu, V. Naresh Boddeti, and M. Savvides, "Local binary convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4284–4293.

[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning*, 2010, pp. 807–814.

[39] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *CoRR abs/1503.02531*, 2015, pp. 1–9.

[40] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[41] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proceedings of the European Conference on Computer Vision*, 2014, pp. 740–755.

[42] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[43] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 82–92.

[44] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *Proceedings of the IEEE International Conference on Computer Vision*, 2011, pp. 991–998.

[45] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

[46] F. Juefei-Xu, V. N. Boddeti, and M. Savvides, "Perturbative neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3310–3318.

[47] W. Chen, D. Xie, Y. Zhang, and S. Pu, "All you need is a few shifts: Designing efficient convolutional neural networks for image classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7241–7250.

[48] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2971–2989, 2021.

[49] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. D. Goodman, W. Banzhaf, and V. N. Boddeti, "Multiobjective evolutionary design of deep convolutional neural networks for image classification," *IEEE*

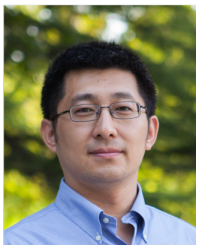
Transactions on Evolutionary Computation, vol. 25, no. 2, pp. 277–291, 2021.
[50] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017, pp. 936–944.
[51] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *Proceedings of the International Conference on Learning Representations*, 2019, pp. 1–16.
[52] S. S. Gill, M. Xu, C. Ottaviani, P. Patros, R. Bahsoon, A. Shaghaghi, M. Golec, V. Stankovski, H. Wu, A. Abraham, M. Singh, H. Mehta, S. K. Ghosh, T. Baker, A. K. Parlikad, H. Lutfiyya, S. S. Kanhere, R. Sakellariou, S. Dustdar, O. Rana, I. Brandic, and S. Uhlig, "Ai for next generation computing: Emerging trends and future directions," *Internet of Things*, vol. 19, no. 2, pp. 1–34, 2022.

Chuntao Ding is currently a lecturer at Beijing Jiaotong University. He received his Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2021. He also worked as a research assistant at HongKong Polytechnic University and as a visiting scholar at Michigan State University. His research interests include Cloud Computing and Deep Learning. His research papers were published in many prestigious journals and conferences including IEEE TMC, IEEE ICWS.

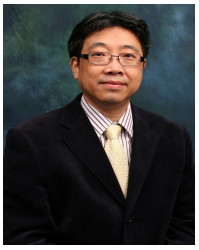
Zhichao Lu received his Ph.D. degree from Michigan State University, USA, in 2020. He is currently an Assistant Professor at Sun Yat-sen University. His research interests include machine learning assisted evolutionary algorithms, automated machine learning, and in particular evolutionary neural architecture search. He received the Best Paper Award at GECCO 2019 under evolutionary machine learning track.

Felix Juefei-Xu received the Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University (CMU), Pittsburgh, PA, USA. Prior to that, he received the M.S. degree in Electrical and Computer Engineering and the M.S. degree in Machine Learning from CMU, and the B.S. degree in Electronic Engineering from Shanghai Jiao Tong University (SJTU), Shanghai, China. Currently, he is a Research Scientist with Alibaba Group, Sunnyvale, CA, USA, with research focus on a fuller understanding of deep learning where he is actively exploring new methods in deep learning that are statistically efficient and adversarially robust. He also has broader interests in pattern recognition, computer vision, machine learning, optimization, statistics, compressive sensing, and image processing. He is the recipient of multiple best/distinguished paper awards, including IJCB'11, BTAS'15-16, ASE'18, and ACCV'18.

Vishnu Naresh Boddeti is an Assistant Professor in the computer science department at Michigan State University. He received a Ph.D. degree in Electrical and Computer Engineering program at Carnegie Mellon University in 2013. His research interests are in Computer Vision, Pattern Recognition and Machine Learning. He received the best paper award at BTAS 2013, the best student paper award at ACCV 2018, and the best paper award at GECCO 2019.



Yidong Li is a professor in the School of Computer and Information Technology at Beijing Jiaotong University. Dr. Li received his B.Eng. degree in electrical and electronic engineering from Beijing Jiaotong University in 2003, and M.Sci. and Ph.D. degrees in computer science from the University of Adelaide, in 2006 and 2010, respectively. Dr. Li's research interests include big data analysis, privacy preserving and information security and intelligent transportation. Dr. Li has published more than 150 research papers in various journals and refereed conferences. He has organized several international conferences and workshops and has also served as a program committee member for several major international conferences.



Jiannong Cao received his M.Sc. and Ph.D. degrees in computer science from Washington State University. He is currently a Chair Professor with the Department of Computing at The Hong Kong Polytechnic University (PolyU). He is also the dean of Graduate School, the director of Research Institute of Artificial Intelligent of Things, the director of the Internet and Mobile Computing Lab and the vice director of the University's Research Facility in Big Data Analytics in PolyU. His research interests include distributed systems and blockchain, wireless sensing and networking, big data and machine learning, and mobile cloud and edge computing. He has co-authored 5 books, co-edited 9 books, and published over 500 papers in major international journals and conference proceedings. He is a member of Academia Europaea, fellow of IEEE, and an ACM distinguished member.