Transforming Large-Size to Lightweight Deep Neural Networks for IoT Applications

RAHUL MISHRA and HARI GUPTA, Indian Institute of Technology (Banaras Hindu University) Varanasi

Deep Neural Networks (DNNs) have gained unprecedented popularity due to their high-order performance and automated feature extraction capability. This has encouraged researchers to incorporate DNN in different Internet of Things (IoT) applications in recent years. However, the colossal requirement of computation, energy, and storage of DNNs make their deployment prohibitive on resource-constrained IoT devices. Therefore, several compression techniques have been proposed in recent years to reduce the energy, storage, and computation requirements of the DNN. These techniques have utilized a different perspective for compressing a DNN with minimal accuracy compromise. This encourages us to comprehensively overview DNN compression techniques for the IoT. This article presents a comprehensive overview of existing literature on compressing the DNN that reduces energy consumption, storage, and computation requirements for IoT applications. We divide the existing approaches into five broad categories—network pruning, sparse representation, bits precision, knowledge distillation, and miscellaneous—based upon the mechanism incorporated for compressing the DNN. The article discusses the challenges associated with each category of DNN compression techniques and presents some prominent applications using IoT in conjunction with a compressed DNN. Finally, we provide a quick summary of existing work under each category with the future direction in DNN compression.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Human-centered computing** → **Ubiquitous and mobile devices;**

Additional Key Words and Phrases: Compression, knowledge distillation, pruning, sparse representation

ACM Reference format:
Rahul Mishra and Hari Gupta. 2023. Transforming Large-Size to Lightweight Deep Neural Networks for IoT Applications. *ACM Comput. Surv.* 55, 11, Article 233 (February 2023), 35 pages.
https://doi.org/10.1145/3570955

1 INTRODUCTION

Recent years have witnessed significant growth in **Internet of Things (IoT)** applications due to easier and cheaper availability of small-sized sensors [1]. The sensors in the IoT generate a huge amount of sensory data that can be used for training machine learning and deep learning models. These models build a classifier by learning a mapping between sensory instances and their corresponding labels. The built classifiers predict several activities and tasks of different domains,

Authors' address: R. Mishra and H. Gupta, Department of Computer Science and Engineering, Indian Institute of Technology (Banaras Hindu University) Varanasi, Varanasi, U.P., India, 221005; emails: {rahulmishra.rs.cse17, hariprabhat.cse}@iitbhu.ac.in.

0360-0300/2023/02-ART233 $15.00
https://doi.org/10.1145/3570955

including agriculture, education, healthcare, passenger assistance, transportation, and tourism, among others [2, 3]. The machine learning based approaches require handcrafted features, which reduce their suitability for a new dataset, where information about data distribution is missing [4]. Therefore, **Deep Neural Network (DNN)**-based IoT applications have achieved dominance in the past decade [5–10]. This is because the DNN provides automatic feature extraction capability from a large amount of raw sensory data. A DNN not only eliminates the human intervention for calculating domain-related features but also achieves coercive performance [5]. A DNN involves the following components, including a **Convolutional Neural Network (CNN)**, **Fully Connected (FC)** layers, and a **Recurrent Neural Network (RNN)**. A DNN can have all of these components at a time or any of its combinations. The CNN extracts spatial features, and the RNN identifies temporal features from the dataset. These features achieve high-order performance in classification, regression, and prediction [11–13].

Despite these advantages, the DNN requires a significant amount of resources, including energy, processing capacity, and storage. This resource requirement reduces the suitability of a DNN for **Resource-Constrained Devices (RCDs)** in IoT [14, 15]. The huge resource requirements of the DNN also become a bottleneck for real-time inference on RCDs and browser-based applications. Therefore, to mitigate these shortcomings of DNNs (i.e., energy, processing capacity, and storage), different DNN compression techniques have been proposed in the existing literature [16–32]. We can achieve several benefits by compressing the DNN over the traditional cumbersome DNN:

- *Storage capacity*: A DNN achieves significant accuracy that comes up with a large number of parameters, requiring considerable storage [33, 34]. Therefore, by compressing the DNN, we can preserve storage that facilitates the deployment of the DNN on RCDs.
- *Computation requirements*: A large number of **FLoating point OPerations (FLOPs)** involved in the DNN operation can exceed the limited computational capacity of the RCDs [35]. Thus, it would be beneficial to employ DNN compression for reducing the computation requirement.
- *Earliness*: The training and inference time of a DNN is significantly high, which hampers its real-time inference performance [36, 37]. Therefore, DNN compression mechanisms provide a higher degree of earliness in both training and inference phases.
- *Privacy*: The data transmission from the source to the high-end machine leads to security breach and privacy compromise [38]. Thus, it would be beneficial to employ in situ processing using a compressed DNN on RCDs, which helps in preserving privacy and data security.

The demand for IoT-based applications is growing day by day, encouraging data scientists to incorporate DNNs in IoT applications. Therefore, it is beneficial to provide a thorough overview of the DNN compression technique that can meet the limited storage and processing capacity available at the resource-constrained IoT devices. Thus far, efforts have been made in the existing literature to provide the surveys on DNN compression techniques [39–41]. For example, Cheng et al. [39] have covered the overview of DNN compression techniques, including parameters pruning and quantization, low-rank factorization, and knowledge distillation. However, the authors missed some crucial DNN compression techniques, such as layer pruning, weight sharing, estimation using integer, teaching assistant, and domain adaptation in knowledge distillation.

Furthermore, Nan et al. [41] surveyed the layer compression techniques for DNNs, including weight factorization and pruning, convolutional decomposition, and unique layer design. However, the survey has covered the shortcomings of including layer compression but cannot summarize the vital technique of knowledge distillation for improving the performance of compressed DNNs. Similarly, Deng et al. [40] covered the DNN compression techniques emphasizing on co-designing of the hardware and algorithm. Here, the authors surveyed hardware-specific compression for

achieving an optimal lightweight DNN. The survey only covers algorithmic compression such as tensor decomposition, data quantization, and sparsification.

Thus, we carry out a comprehensive survey of the existing literature on different DNN compression techniques and propose a helpful categorization to point out a potential research gap for future work. Apart from the existing surveys, we summarize all of the novel approaches in the literature on DNN compression such that the reader can get the overview from a single source. In addition, we try to mitigate all of the shortcomings of the existing surveys on DNN compression techniques such that researchers could benefit when developing lightweight DNNs for limited-resource IoT devices.

The rest of the article is structured as follows. Section 2 discusses the categorization of DNN compression. Section 3 presents an overview of network pruning. The sparse representation techniques are presented in Section 4. Further, Sections 5 and 6 demonstrate the detailed description of bits precision and knowledge distillation techniques, respectively. Miscellaneous techniques are discussed in Section 7. Section 8 illustrates different applications of IoT using compressed DNNs. Finally, Section 9 presents discussions and future directions.

2 CATEGORIZATION OF DNN COMPRESSION TECHNIQUES FOR IOT

The existing work [16–32] on DNN-based recognition using sensory data adopts different mechanisms to enhance the deployment of DNNs on RCDs. These mechanisms not only preserve data privacy due to local computation but also decrease the energy consumption during computation. We categorize the existing work on DNN compression into five broad categories—network pruning, sparse representation, bits precision, knowledge distillation, and miscellaneous techniques—for DNN compression. These categories were chosen through an extensive literature review on sensory data based DNN compression techniques. Figure 1 illustrates an overview of the existing work under different categories of DNN compression for IoT.

2.1 Network Pruning

Deep network pruning is one of the popular techniques to reduce the size of a deep learning model by incorporating the removal of the inadequate components such as channels, filters, neurons, or layers to produce a lightweight DNN [42, 43]. The resultant lightweight DNN has low power consumption, is memory efficient, and provides faster inference with minimal accuracy compromise. The adequacy of a component relies on the amount of loss incurred when a component is removed from the DNN [31, 44]. We further categorized the network pruning techniques into four parts: channel pruning, filter pruning, connection pruning, and layer pruning. An overview of existing network pruning techniques [16, 31, 45–64] is presented in Section 3.

2.2 Sparse Representation

Sparse representation exploits the sparsity present in the weight matrices of the DNN [65]. In sparse representation, the weights that are zero or near to zero are removed from the weight matrix, which reduces the storage and computation requirements of the DNN. In other words, those links in the network having similar weights are multiplexed, where *multiple weights with a single link* is replaced with a *single weight with the multiplex link*. The sparse representation includes low-rank estimates [66] and multiplexing, among others. The principal motive behind the sparse representation is to minimise the number of bits required to store weight matrix in memory without loosing the performance. In this work, we divide the DNN compression using sparse representation into two subcategories: multiplexing, and weight sharing. Section 4 presents a detailed description of the sparse representation techniques in the literature [17, 18, 56, 62, 67–70] on DNN compression.

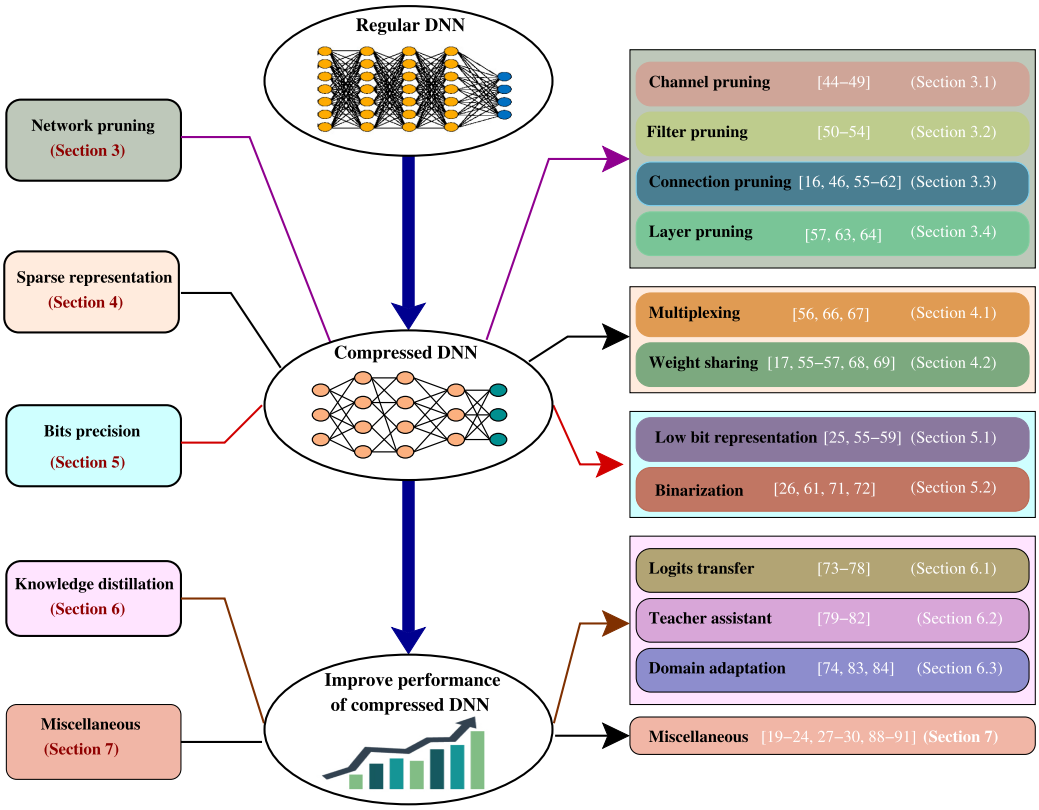

Fig. 1. Overview of different categories of compression techniques for DNNs.

2.3 Bits Precision

Bits precision in the DNN compression technique reduces the number of bits required for storing the weights in the weight matrices. We categorized the existing literature on DNN compression using bits precision into two subcategories: low bits representation and binarization. Section 5 highlights the overview of different bits precision techniques [25, 26, 55, 57, 60, 64, 71, 71–74] for DNN compression using a limited number of bits.

2.4 Knowledge Distillation

In the DNN, the term *knowledge distillation* is defined as the process of transferring the generalization ability of the cumbersome DNN (teacher) to the compact DNN (student) to improve its performance [32, 75]. Knowledge distillation provides a mechanism to overcome the accuracy compromise due to DNN compression. The training of the student using knowledge distillation improves its generalization ability such that it can mimic behavior of the teacher for predicting the class label probabilities. In this work, we present a detailed overview of three categories of knowledge distillation: logits transfer, teacher assistant, and domain adaptation. Section 6 provides a description of the all subcategories of knowledge distillation in the literature [75–87].

2.5 Miscellaneous

The DNN compression techniques that do not qualify for the preceding four categories are classified as miscellaneous. It includes such compression techniques, which perform modeling

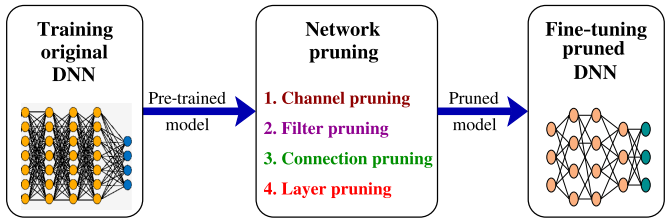

Fig. 2. Categories and steps involved in network pruning.

of DNN in such a manner that can be easily fitted on mobile devices. The existing literature [19–24, 27–30, 88–95] reflecting the miscellaneous category is discussed in Section 7.

3 NETWORK PRUNING

The two major research challenges associated with network pruning are mitigating the accuracy compromise due to network compression and reducing the time in fine-tuning of the DNN [96, 97]. We divide network pruning into four categories: channel pruning [45–49, 98], filter pruning [50–54], connection pruning [16, 31, 55–62, 98], and layer pruning [62–64]. Finally, we summarize the lottery ticket hypothesis. Figure 2 illustrate the categories and steps involved in network pruning. Table 1 summarizes prior studies on network pruning.

3.1 Channel Pruning

In this section, we provide an overview of different channel pruning techniques discussed in the existing literature. Channel pruning is a concept of reducing the number of channels in the input supplied to the intermediate layers of the DNN [99]. The data provided to the DNN are initially channelized to form an appropriate input. Each layer of the DNN also contains different channels that improve performance but increase the computation and storage. He et al. [45] proposed an inference time approach to perform pruning by effectively removing the redundant channels.

Let k represent the number of channels in a feature map that is pruned to reduce the channels to k'. Let a convolutional filter $\mathbf{F}$ of size $k \times k \times f_h \times f_w$ be applied to an input $\mathbf{I}$ of dimension $N \times k \times f_h \times f_w$ to produce output matrix $\mathbf{O}$. N denotes the number of samples in dataset $\mathcal{D}$, and $f_h \times f_w$ is the dimension of the convolutional kernel. The minimization problem is formulated as follows:

$$\arg\min_{\delta,\mathbf{F}} \frac{1}{N}\left\|\mathbf{O} - \sum_{j=1}^{k} \delta_j \mathbf{F}_j \mathbf{I}_j\right\|_F^2, \text{ subject to, } ||\delta||_0 \leq k', \tag{1}$$

where $||\cdot||_F$ denotes the Frobenius norm, $\mathbf{I}_j$ represents the input for channel j, $1 \leq j \leq k$, $\mathbf{F}_j$ is the j^{th} slice of $\mathbf{F}$, δ is a coefficient vector of length k used in channel selection, and δ_j is the j^{th} entry of δ.

Liu and Liu [46] proposed an improved scheme of pruning operation where at the initial step, the pruning threshold is determined by analyzing the network connections. Further, those connections and channels are removed that are less than the pruning threshold. The threshold estimation in the proposed scheme preserves the accuracy of the pruned network up to some extent. Chen et al. [98] claim that the prior assumption of channels and neurons independent in the DNN is wrong. The authors formed a combinatorial optimization problem for network pruning that is solved using an evolutionary algorithm and attention mechanism. They named the proposed mechanism **Evolutionary NetArchitecture Search (EvoNAS)**.

Table 1. Summary of Network Pruning Techniques for DNN Compression

Paper	Address the Challenge	Proposed Solution	Suitable for			Type
			CNN	FC	RNN	
[45]	To reduce accumulate errors and enhance compatibility with DNN architectures	Efficiently remove redundant channels while maintaining accuracy	✓	✗	✗	Channel pruning
[46]	To improve channel pruning for making a compressed DNN achieve the pruning limit	Determine the threshold before actual pruning to match the speed of embedded devices	✓	✗	✗	
[98]	To prune by searching an appropriate architecture; to resolve the dependency among the channels or neurons during pruning	Combine an evolutionary algorithm and attention mechanism to solve the pruning problem via evolutionary architecture search	✓	✓	✗	
[47]	To reduce size, decrease inference time, and curtail computational operations with no performance deterioration	Automatically prune the insignificant channels during training ; network slimming by exploiting channel-level sparsity	✓	✗	✗	
[48]	To handle conflict between limited and unbalanced computational intensities of the DNN for tiny devices	Employ an equivalence of angular loss and variable group convolution to guide a lightweight DNN	✓	✗	✗	
[49]	To build smaller and faster MobileNets by resolving the tradeoff between reasonable performance and size of the DNN	Hyperparameters to manage the tradeoff between latency and performance	✓	✗	✗	
[50]	Efficient framework for accelerating operations	Discard unimportant filters using statistical information	✓	✗	✗	Filter pruning
[51]	To reduce the storage and memory during training and inference	Sparsification in the FC layer and convolutional filters	✓	✓	✗	
[52]	Prune parameters to minimize degradation	Knee-guided evolutionary algorithm for optimal filter pruning	✓	✗	✗	
[16]	To ensure the deployment of the DNN on IoT devices	Optimal dropout estimation	✓	✓	✓	Connection pruning
[31]	To handle uncertainty of inference on RCDs	Optimal dropout and predictive estimation of sensory data distribution	✓	✓	✓	
[55]	To reduce storage and bandwidth of the DNN with no accuracy compromise	Quantization using a clustering technique and further Huffman coding	✓	✓	✓	
[56]	To perform DNN-based inference on the compressed network	Prune redundant connections and weight sharing among elements	✓	✓	✓	
[62]	To induce a minimal error on layer sharing	Layer-wise neuron sharing and subsequent weight update	✓	✓	✓	
[58]	To improve performance and provide energy efficiency	0-valued weight estimation generated from ReLU	✓	✗	✗	
[59]	Sparse solution for compressing the DNN	Use sparse variational dropout	✓	✗	✗	
[63]	To build a fast and flexible heterogeneous DNN	Optimization for mapping a Binary Neural Network (BNN) to hardware	✓	✓	✗	Layer pruning
[64]	Pruning method to deploy the DNN on embedded devices	Singular value decomposition based factorization	✗	✗	✓	
[62]	Minimal change in error upon layer sharing	Layer pruning and weight update	✓	✓	✓	

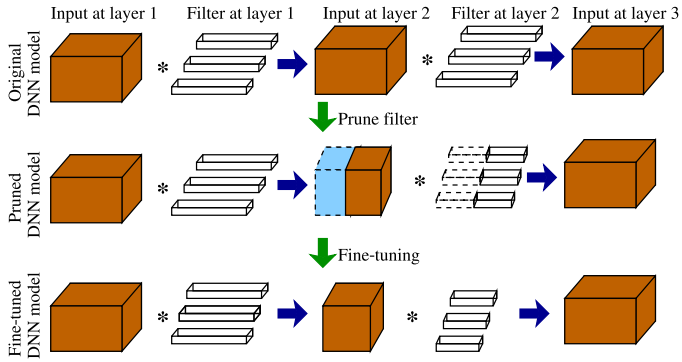

Fig. 3. An example scenario of the filter-level pruning [50].

Liu et al. [47] proposed a novel learning scheme for the CNN that decreases the size, runtime memory, and computing operations. The proposed scheme enforces channel-level sparsity in the network and directly applies to CNN architectures. Therefore, it introduces minimal overhead during the training of the deep learning model. This approach's main idea is to introduce a scaling factor (Δ) for each channel in the deep learning model. Yan et al. [48] proposed an approach (named *VarGNet*) for reducing the operations in the CNN that uses variable group convolution. The use of variable group convolution helps in solving the conflict between small computational cost and the unbalanced computational intensity. Howard et al. [49] proposed a DNN compression technique based on a streamlined architecture that uses depthwise separable convolutions to build a lightweight DNN. The authors named the technique *MobileNets*, which estimates two hyperparameters: width multiplier and resolution multiplier. In MobileNets, the depthwise convolution applies a single filter to each input channels. Further, 1×1 convolution is performed to combine inputs into a new set of output. The role of the width multiplier wd is to thin a network uniformly at each layer. For a given layer, the number of input channels M becomes $(wd)M$.

Remarks. The existing literature on EvoNAS [98], VarGNet [48], MobileNets [49], and others [45–47] covered different mechanisms of channel pruning. However, none of the existing work helped in diminishing the accuracy compromise due to channel pruning. Apart from the accuracy compromise, the existing literature missed channel elimination on the performance of different layers of the DNN, as the removal of a single channel in one layer can also affect other layers in the DNN.

3.2 Filter Pruning

The convolutional operation in the CNN incorporates a large number of filters to improve its performance under different processes of classifications, regressions, and predictions. It is assumed that the increment in the number of filters improves the distinguishable characteristics of the spatial features generated from the CNN [100]. However, this increment in the convolutional filters leads to a significant increase in the number of FLOPs in the DNN. Therefore, eliminating unimportant filters plays a decisive role in reducing the computational requirements of the DNN. An example scenario of filter-level pruning is illustrated in Figure 3.

Luo et al. [50] proposed an efficient framework named *ThiNet* for accelerating the operation of a CNN using compression in both training and testing phases. They incorporated filter-level pruning, where an unimportant filter is discarded based on the statistical information computed from the next layer. Luo et al. [50] also established filter-level pruning as an optimization problem

to determine the filters that are to be pruned:

$$\arg\min_{E} \sum_{i=1}^{N} \left(\hat{y}_i - \sum_{j \in E} \hat{\mathbf{x}}_{ij} \right)^2, \text{ subject to, } |E| = k \times c_{rate}, \quad E \subset \{1, 2, \ldots, k\}, \tag{2}$$

where N is the number of training examples $(\hat{\mathbf{x}}_i, \hat{y}_i)$, $|E|$ is the number of elements in a subset of E, k denotes number of channels in the CNN, and c_{rate} is compression rate that determines the number of channels retrained after compression.

Bhattacharya and Lane [51] proposed a DNN compression technique that removes the sparsification in an FC layer and convolutional filters. The main motive was to reduce the storage requirement of device and processing resources in both training and inference phases. The basic operational principle of this work is the hypothesis that the computational and space complexity of the DNN can be considerably decreased using sparse layers and filters.

Zhou et al. [52] defined filter pruning as a multi-objective optimization problem followed by a knee-guided evolutionary algorithm for its solution. They established a tradeoff between the number of parameters and performance compromise due to compression. The basic principle of this mechanism is to prune parameters that result in little performance degradation. To estimate the importance of a parameter, they used the criteria of performance loss to identify redundancy. To achieve a compressed DNN with a small size, the number of filters preserved must be as few as possible to achieve considerable accuracy. Huynh et al. [53] proposed a mechanism named *DeepMon* to provide deep learning inference on mobile devices. They claimed to run the inference quickly and provided energy efficiency using the **Graphical Processing Unit (GPU)** on the mobile device. The authors proposed an optimization mechanism for processing convolutional operations on mobile GPU. The mechanism utilizes the internal processing structure of the CNN, incorporating filters and the number of connections to reuse the results. Thus, removing unimportant filters and connections leads to faster inference.

Denton et al. [54] speed up the test-time evaluation of a large DNN, designed for object recognition tasks. The authors exploited the redundancy present within the convolutional filters to derive approximations that significantly reduce the required computation. They started compression of each convolutional layer by finding an appropriate low-rank approximation and then fine-tuned it.

Remarks. The filter pruning technique presented in the existing literature [50–54] has successfully reduced the number of FLOPs in the CNN. However, the FC layers and recurrent layers also contribute a major portion of FLOPs in the DNN, which should not be ignored during DNN compression. In addition, the existing work cannot be employed on a DNN with a large number of layers with minimal filters on each layer. This induces the need for future work to compress the long-sized horizontal network.

To specify the difference between filter and channel pruning, let us consider a **General Matrix Multiplication (GEMM)** matrix having k filters and m channels, as shown in Figure 4(b). GEMM can be obtained by transforming weight and feature maps tensors [101]. In the GEMM matrix, filter pruning corresponds to reducing one or more row(s); thus, it is termed *row pruning*. Channel pruning corresponds to the elimination of one or more column(s) for the GEMM matrix; hence, it is also called *column pruning*. Although filter and channel pruning are different, a correlation exists between them—that is, pruning a filter at the i^{th} layer is similar to pruning the corresponding channel at the $(i+1)^{th}$ channel, generated by the specific filters, as shown in Figure 4(a).

3.3 Connection Pruning

The number of input and output connections to a layer of a DNN determines the number of parameters. These parameters can be used to estimate the storage and computation requirement of

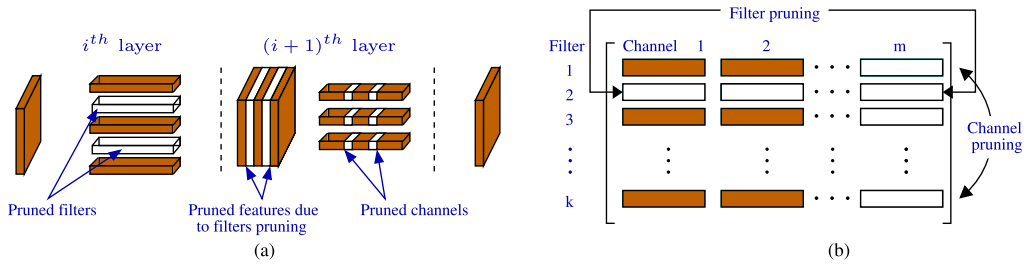

Fig. 4. Filter pruning and channel pruning. (a) Illustration of the relation between the filter and channel pruning. Filter pruning at the i^{th} layer eliminates the feature maps, generating inactive channels to be pruned at the $(i+1)^{th}$ layer. (b) GEMM matrix [101].

the DNN [95]. Since the DNN requires a large number of parameters in its operations, it is convenient to reduce the parameters by eliminating unimportant connections for different layers in the DNN [102]. The existing studies on connection pruning [16, 31, 55–62, 98] have attempted to remove the unimportant connection from the DNN.

Yao et al. [16] proposed an approach that helps in the deployment of deep learning models on IoT devices. Thus, they named the approach *DeepIoT*. The proposed approach can perform compression on commonly used deep learning structures, including CNNs, FC layers, and RNNs. DeepIoT incorporates dropout techniques to remove unimportant connections from the network. Here, the chosen dropout is not a fixed value but is optimally determined for generating optimal compressed networks. In DeepIoT, small dense matrices are obtained by performing compression of the different structures in the network. These dense matrices contain a minimal number of non-redundant elements, such as the input and output dimension of a layer and filters, among others. The process of obtaining dense matrices must safeguard the network from higher accuracy compromise. Yao et al. [31] proposed a scheme named *RDeepSense* that incorporates the dropout of unimportant connections from a DNN. In addition, it uses a predictive estimation of the sensory data distribution to handle the uncertainty of inference on RCDs. The authors claim to reduce energy consumption by around 90%.

Han et al. [55] proposed a connection pruning technique that first prunes the DNN by learning only important connections using dropout. Next, the learned weights are quantized using the clustering technique, and further Huffman coding is applied to reduce the storage requirement. The authors under this technique try to fit the DNN into cache memory rather than main memory. In this way, they encourage us to run applications on tiny computing devices with limited storage.

Han et al. [56] proposed an **Energy-efficient Inference Engine (EIE)** to perform DNN-based inference on the compressed network obtained after pruning of redundant connections and weight sharing among different elements. The EIE accelerates the sparse matrix multiplication by adopting weight sharing without losing the accuracy. In EIE, the processing is performed using an array of processing elements. Each element simultaneously partitions the network in SRAM and performs the computation of its respective part. He et al. [62] proposed a **Multi-Tasking Zipping (MTZ)** framework that automatically merges multiple correlated DNNs to perform cross-model compression. In MTZ, the compression is performed using layer-wise neuron sharing and subsequent weight updates. Here, the authors induced a minimal change in error upon layer sharing.

Qin et al. [57] developed a quantitative characterization approach for a deep learning model to make its deployment on embedded devices. The approach provides a better selection mechanism for the deployment of a DNN on the embedded device and guides the development of

compression techniques using a wiser connection pruning. Parashar et al. [58] proposed a zero-valued weight estimation from a network during training that is generated from ReLU operation. The authors named the approach **Sparse CNN (SCNN)**, which is a DNN compression architecture that improves performance and provides energy efficiency. SCNN uses both weight and activation sparsity.

Molchanov et al. [59] proposed a sparse variational dropout, which is an extension of variational dropout. Here, all possible values of dropouts are considered, which leads to a sparse solution for compressing the DNN. The authors provided an approximation technique that incorporates **Kullback-Leibler (KL)** divergence in variational dropout. They experimentally demonstrate that higher sparsity is achieved in the CNN and FC layers. Let N denote the number of instances in dataset $\mathcal{D}$, whose i^{th} instance is denoted as $(\mathbf{x}_i, y_i)_{i=1}^{N}$. The goal of a learning mechanism is to estimate parameter θ of a model $p(y/\mathbf{x}, \theta)$. When we incorporate Bayesian learning after data $\mathcal{D}$ arrival, the prior distribution is transformed into a posterior distribution, $p(\theta/\mathcal{D}) = p(\mathcal{D}/\theta)p(\theta)/p(\mathcal{D})$. In variational inference, the posterior distribution $p(\theta/\mathcal{D})$ is approximation using distribution $r_\phi(\theta)$. The authors uses KL divergence $D_{KL}(r_\phi(\theta)||p(\theta/\mathcal{D}))$ to estimate quality of approximation. The optimal value of parameter ϕ is obtained as follows:

$$\mathcal{L}(\phi) = \mathcal{L}_D(\phi) - D_{KL}(r_\phi(\theta)||p(\theta)), \text{where } \mathcal{L}_D(\phi) = \sum_{i=1}^{N} \mathbb{E}_{r_\phi(\theta)}[\log p(y_i/\mathbf{x}_i, \theta)].$$

Louizos et al. [60] proposed a Bayesian compression for deep learning. They introduced a hierarchical priority to prune the nodes inside the DNN despite weight reduction. The authors incorporated posterior uncertainties for determining the fixed-point precision to encode the weights. Further, the authors employed inducing sparsity for hidden neurons despite individual weights to prune neurons. Han et al. [103] proposed a network pruning method for the DNN that removes unimportant connections. The proposed pruning method first identifies the unimportant connections having weights less than a given threshold. Next, these connections are removed from the DNN. Further, fine-tuning is performed.

Guo et al. [61] proposed a network compression technique to perform network pruning. The pruning incorporated deletion of unimportant parameters and retaining the remaining ones. The authors incorporated splicing into compression to avoid incorrect pruning. There are two major issues related to network interconnections in a DNN. The first issue is irretrievable network damage due to incorrect pruning, and the second is the inconsistency of the DNN due to inefficient machines.

Remarks. The connection pruning technique provides a high-order compression of the DNN in contrast with channel pruning and filter pruning discussed in Sections 3.1 and 3.2, respectively. The existing studies on RDeepSense [31], DeepIoT [16], EIE [56], MTZ [62], SCNN [58], and others [55, 57, 59–61, 98] have emphasized on connection pruning to perform DNN compression. However, a major concern of the connection pruning technique is the elimination of an important connection during DNN compression and needs more emphasis, such as pruning and splicing discussed in the work of Guo et al. [61]. In addition, connection pruning suffers from higher accuracy compromise when the network is dynamically built and rebuilt. It provides the future direction to develop the mechanisms that curtail the need for frequent training and retraining while transforming large-size DNNs into lightweight ones.

3.4 Layer Pruning

The last category of the network pruning techniques is layer pruning, where some selected layers from the network are removed to compress the DNN. Layer pruning is highly utilized for deploying

a DNN on tiny computing devices, where we need an ultra-high compression of the DNN [104]. The primary issue with layer pruning is the loss of the semantic structure of the DNN that generates low-quality features. These low-quality features result in performance inefficiency.

Tan et al. [63] presented a framework that builds fast and flexible heterogeneous architecture, named *FINN*. They utilized a set of optimization for mapping binarized neural networks to the hardware. The authors implemented different DNN components, including convolutional, pooling, and FC layers. The implementation was conducted in a manner that meets the performance demand of the users. Chauhan et al. [64] compared the performance of different classification approaches and proposed a pruning method for deploying DNN on embedded devices. They took three embedded devices, including smartphones, smartwatches, and Raspberry Pi.

Louizos et al. [60] have used the Singular Value Decomposition (SVD)-based factorization method that allows the decomposition of weight matrix ($\mathbf{W}$) into three submatrices: $\mathbf{A}$, $\mathbf{E}$, and $\mathbf{B}$ of dimensions $N \times l$, $l \times l$, and $l \times M$, respectively. Here, N denotes the number of instances in dataset $\mathcal{D}$ having sample length of size M, l denotes the number of class labels in the dataset, and matrix $\mathbf{E}$ is a diagonal matrix. The decomposition of the weight matrix is represented as follows:

$$\mathbf{W}_{N\times M} = \mathbf{A}_{N\times l}\mathbf{E}_{l\times l}\mathbf{B}_{l\times M}. \tag{3}$$

Weight factorization helps in achieving both storage gain $\mathcal{S}_g$ and computation gain $\mathcal{C}_g$ defined using the following expressions:

$$\mathcal{S}_g = \frac{N \times M}{N \times l + l^2 + l \times M} \text{ and } \mathbf{C}_g = \frac{N^2 \times M}{N^2 \times l + l^3 + l^2 \times M}. \tag{4}$$

Remarks. The existing literature on layer pruning, FINN [63], and others [62, 64] has reduced both the storage and computation requirement of the DNN by providing an ultra-high-level pruning. However, layer pruning results in higher accuracy compromise due to structural deterioration of the DNN, which should be mitigated to enhance the utility of layer pruning in the future.

Network pruning helps reduce the size of the DNN by removing unimportant channels, filters, connections, and layers. These removals generate a small-sized DNN, which makes training more efficient in accordance with the time and resources. Later, the small-sized DNN can be easily deployed and trained on the RCDs to generate results during testing. Network pruning determines multiple small-sized networks from large-sized DNNs using a different combination of channels, filters, connections, and layers. Thus, it is tedious to train multiple small-sized DNNs from the start with randomly initialized parameters [105]. In addition, random initialization also compromises the achieved performance. To cope with such a challenge, Frankle and Carbin [105] introduced the concept of *lottery ticket hypothesis*. The hypothesis determined a dense subnetwork (winning ticket) from a large-sized DNN, where the subnetwork achieved performance similar to the large-sized DNN when trained in isolation for a nearly equal number of epochs.

Let us assume a large-sized DNN, denoted as $\mathcal{F}(\mathbf{X}; \mathbf{W})$, having initial weight parameter $\mathbf{W} = \mathbf{W}_o$ and input $\mathbf{X}$. During training, $\mathcal{F}(\mathbf{X}; \mathbf{W})$ is optimized using stochastic gradient descent on a training set to achieve minimum loss $\mathcal{L}$ and test accuracy α at epoch e. Further, assuming the training of $\mathcal{F}(\mathbf{X}; \Omega; \mathbf{W})$ with a fixed mask $\Omega \in \{0, 1\}^{\|\mathbf{W}\|}$ on the parameter $\mathbf{W}$, where the DNN is initialized using $\Omega \odot \mathbf{W}_o$. Now, optimizing $\mathcal{F}(\mathbf{X}; \Omega; \mathbf{W})$ on the same training set using stochastic gradient descent, $\mathcal{F}(\cdot)$ reached validation loss $\mathcal{L}'$ to achieve accuracy α' at epoch e'. The lottery ticket hypothesis predicts the suboptimal value of Ω at given $e' \leq e$ (less or equal training epochs), $\alpha' \geq \alpha$ (greater or equal accuracy), and $\|\Omega\| << \|\mathbf{W}\|$ (much fewer parameters). The steps involved in the lottery ticket hypothesis to determine the winning ticket (suboptimal Ω) [105] are as follows:

(1) At the initial step, a large-sized DNN ($\mathcal{F}(\mathbf{X};\mathbf{W})$) is randomly initialized, where $\mathbf{W} = \mathbf{W}_o$.
(2) Next, the large-sized DNN $\mathcal{F}(\mathbf{X};\mathbf{W})$ is trained for e epochs to obtain the parameter $\mathbf{W}_e$.
(3) Further, pruning is performed to remove $p\%$ of the parameters from $\mathbf{W}_e$ via a mask Ω.
(4) Finally, the remaining parameters are reset to their initial values in $\mathbf{W}_o$. This results in the winning ticket $\mathcal{F}(\mathbf{X};\Omega;\mathbf{W}_o)$.

Researchers exploited the effectiveness of the lottery ticket hypothesis and proposed different modifications and upgrades [106–110]. Extending the concept of the lottery ticket hypothesis, Zhou et al. [106] analyzed three critical components and identified their significance variations without compromising accuracy. The three critical components are mask criteria, mask action for ones, and mask action for zeros. The authors also identified masks that can be used in untrained and randomly initialized DNNs to achieve far better performance than direct training. Frankle et al. [107] analyzed the instability of the subnetworks obtained from large-sized DNNs using the lottery ticket hypothesis. The authors identified that the subnetworks reach the desired accuracy if they are stable to the stochastic gradient descent noise. The noise may occur at the initialization of small subnetworks and early training for the large subnetworks. To reduce the complexity of the computation, the authors used linear mode connectivity. The authors referred to a network as stable if the error did not show frequent increasing or decreasing behavior.

Frankle et al. [108] examined various changes that occurred on the DNN in the initial (or early) training phase. The authors leveraged the lottery ticket hypothesis from another work [105] to identify quantitative distribution and specified their roles in different aspects of the datasets. Through experiments, the authors determined that the weighted distributions are highly non-independent even after only a few training epochs. The authors evaluated the impact of deviation in the networks' state in the initial training stage and obtained that the DNN is not robust against the reinitializations using random weights. You et al. [109] identified the complexity of determining the winning ticket in the lottery ticket hypothesis, as it is costlier to train, prune, and retrain the subnetworks. The authors introduced the mechanism of identifying winning tickets in the early stage of training, namely early-bird tickets, achieved at a higher learning rate. From different experiments, the authors proved that early-bird tickets are consistent. They also introduced the mask distance metric to identify such winning tickets. Aggregately, the authors developed an efficient training framework using early-bird tickets. Similar to You et al. [109], Morcos et al. [110] illustrated the computational expense of determining winning tickets in the lottery ticket hypothesis. The authors proposed a potential solution to reuse the same winning ticket across multiple datasets and optimizers. The authors claimed that reusing one winning ticket performed similarly to dataset-specific initializations through experimental evaluation. Moreover, winning tickets that are generated using a large dataset with many classes are generalized well in contrast to a small dataset with limited classes.

Experimental Outcome. We considered three existing large-sized DNNs—DeepZero [6], DeepFusion [30], and DeepSense [23]—to analyze the impact of DNN compression. All of the considered large-sized DNNs use sensory datasets for training and recognizing locomotion modes or human activities with high accuracy. The sensory datasets are more common and widely used in IoT; therefore, we selected these large-sized DNNs. Table 2 illustrates the impact of convolutional layer pruning on FLOPs and the accuracy of considered large-sized DNNs (DeepZero [6], DeepFusion [30], and DeepSense [23]). We observed the higher accuracy compromise at 80% pruning of convolutional layers. In addition, DeepSense [23] underwent the highest accuracy compromise, as its architecture comprises most of the layers as convolutional.

Let $\mathcal{F}(\mathbf{X},\mathbf{W})$ represent a DNN, where $\mathbf{X}$ and $\mathbf{W}$ denote the input features matrix and weight matrix, respectively. When $\mathcal{F}(\cdot,\cdot)$ is applied on input $\mathbf{X}$, it is transformed layer by layer of the

Table 2. Illustration of Reduction in Accuracy and FLOPs on Compression of Large-Sized DNNs

DNN	Reduction in	Pruning of Convolutional Layers			
		20%	40%	60%	80%
DeepZero [6]	*Accuracy*	03.23%	09.13%	19.38%	42.71%
	FLOPs	19.27%	39.94%	61.98%	85.81%
DeepFusion [30]	*Accuracy*	04.61%	12.37%	23.38%	47.81%
	FLOPs	19.83%	40.21%	62.56%	86.52%
DeepSense [23]	*Accuracy*	6.41%	13.21%	26.53%	51.29%
	FLOPs	18.37%	37.22%	60.32%	84.71%

DNN to generate an output feature vector. Such transformation of $\mathbf{X}$ is performed in the inference stage. In the training stage, this inference stage is called *forward pass*. Apart from the inference stage, the training stage also stores the input of each layer in the weight matrix. In addition to the forward pass, the training stage also comprises the backward pass. The accuracy of the forward pass is estimated using different types of loss functions. Later, the estimated loss function in the forward pass is propagated in the reverse direction to update the weight matrix in the backward pass. These forward and backward passes are orchestrated for a sufficient number of epochs using a large set of data samples to obtain the final weight matrix $\mathbf{W}$. Thus, pruning/ quantization not only accelerates the inference and training stages by reducing the mathematical operations but also reduces the memory to store the weight matrix during the forward pass of the training stage.

4 SPARSE REPRESENTATION

In the previous section, we covered the techniques for DNN compression that prune unimportant components, including layers, filters, and channels, among others. This section gives an overview of the DNN compression techniques that preserve the overall structure of the DNN. The sparsity in the representation of the DNN is exploited to reduce energy, storage, and processing requirements of the DNN [111, 112]. The sparsity in the DNN persists in the weight matrices for the following two reasons:

(1) The value of stored weights is zero or near to zero. It could be beneficial to remove these weights to reduce the computation and storage requirements.
(2) The maximum stored weights are alike, which provides convenience to replace multiple weights having single connections with weight having multiplex connections.

We categorize the existing literature into two parts—multiplexing [56, 67, 68] and weight sharing [17, 18, 55, 56, 62, 69, 70]—as illustrated in Table 3.

4.1 Multiplexing

Multiplexing plays a vital role in reducing the storage requirement of the DNN. The multiplexing is highly effective when weight matrices have similar values of weights [113]. These weights are replaced with a single weight with multiplexed connections. Eshratifar and Pedram [67] proposed a lightweight neural multiplexer. The input to the multiplexer is the raw data and resource budget, which jointly determine the model that should be called to perform inference. The authors introduced a contrastive loss function ($Loss_c$) to train all of the models in the multiplexing by adding $Loss_c$ with the main loss function (e.g., cross-entropy loss) of the model. For a pair of models (M_1 and M_2), three possible cases can happen: both M_1 and M_2 can predict correct output, M_1 or M_2 can predict correct output, and none of them can predict correct output. In the third case, $Loss_c$

Table 3. Illustration of DNN Compression Techniques That Incorporate Sparse Representation

Paper	Address the Challenge	Proposed Solution	Suitable for			Type
			CNN	FC	RNN	
[56]	To perform DNN inference on a compressed network	Prune connections and weight sharing with elements	✓	✓	✓	Multiplexing
[67]	To jointly determine the model to perform inference	Develop a lightweight neural multiplexer	✓	✓	✓	
[68]	Multiplexing recognition and prediction task using a single backbone network	Sequential steps of training a CNN-based backbone network followed by branches	✓	✗	✗	
[17]	To reduce resources of a DNN through decomposition	Layer-wise sparse coding and sparse matrix decomposition	✓	✓	✓	Weight sharing
[18]	To perform DNN inference on the compressed network	Prune redundant connections and weight sharing	✓	✓	✓	
[55]	To reduce energy consumption of a DNN	Energy-preserving DNN compression to reduce storage	✓	✓	✓	
[56]	To perform DNN inference on a compressed DNN	Redundant connections pruning and weight sharing	✓	✓	✓	
[62]	Minimal change in error upon layer sharing	Layer pruning and weight update	✓	✓	✓	
[69]	Data sharing during DNN training	Use a max-margin approach to extract most identifiable data	✓	✓	✓	
[70]	DNN-based modeling framework for RCDs	Follow multi-tasking learning to train a shared DNN	✓	✓	✓	

is not applied, and only the cross-entropy function works. The contrastive loss function ($Loss_c$) is defined as follows:

$$Loss_c = \sum_{i=1}^{N} \sum_{j=1, i\neq j}^{N} \log(dist(a_i, a_j))((\hat{y}_i == y \& \hat{y}_j == y) - (\hat{y}_i ! == y \& \hat{y}_j == y) - (\hat{y}_i == y \& \hat{y}_j == y)),$$

where y and $\hat{y}$ are correct and predicted labels, respectively, and $dist\cdot$ denotes the cosine distance function $dist(a_1, a_2) = \frac{a_1^T a_2}{a_1^T a_1 \times a_2^T a_2}$.

Zhang et al. [68] proposed a multi-branch CNN architecture, which multiplexes different recognition and prediction tasks simultaneously using a single backbone network. The proposed mechanism involved sequential steps. The authors first trained a CNN-based backbone network and then trained different branches of the network by freezing the training of the backbone network. The authors proved the multiplex approach that preserves both time and energy of the system while achieving significant accuracy on an embedded device.

Remarks. The existing literature on multiplexing [56, 67, 68] helps in DNN compression to reduce storage and computation requirements. However, the existing literature designed the framework for multiplexing different DNNs. None of the work tried to multiplex the weights, which can be an effective solution for reducing the storage, as well as a prominent future direction.

4.2 Weight Sharing

In this section, we present an overview of weight sharing in the DNN. Weight sharing proves to be the important criteria for reducing both storage and computation requirements of the DNN [17, 18, 114]. In weight sharing, despite storing multiple weights for the DNN, it would be convenient

to share the pre-existing weights. The weights stored for the $(i-1)^{th}$ layers can be used by the i^{th} layer.

Liu et al. [17] proposed a concept of layer-wise sparse coding for pruning the DNN. They divide the sparse weight matrices into multiple sub-matrices for performing simultaneous execution on RCDs. Lee and Nirjon [18] introduced a compression mechanism that enables a large DNN to be fit smoothly on the main memory of the RCDs. The authors have achieved high-order memory saving on executing a large volume of sensory data. Han et al. [55] used the k-means clustering technique for identifying the shared weights of each layer in the trained network. All connections that are in the same cluster must share the same weights. Here, the authors also assumed that the weights are not shared across the layers in the DNN. They partition m weights in the network, $\mathbf{W} = \{w_1, w_2, \ldots w_m\}$ into c clusters $K = \{k_1, k_2, \ldots k_c\}$, where $m >> c$:

$$\arg\min_{K} \sum_{i=1}^{C} \sum_{w \in k_i} |w - k_i|^2. \tag{5}$$

Wu et al. [69] emphasized the technique of a shared training process that involves sharing of training data. This sharing of data (or weights) provides a better understanding of the prediction process involved in the DNN. However, this data sharing during the model training also leads to privacy compromise. Therefore, the authors proposed a method that discloses a few samples of the training data, which are sufficient for a data analyst to get insight into the DNN. To reduce the disclosure of the training data, the authors used the max-margin approach that extracts the most identifiable training data.

Georgiev et al. [70] proposed a deep learning based modeling and optimization framework for RCDs. The proposed framework achieves considerable accuracy while consuming limited resources. The framework follows a multi-tasking learning principle for training a shared DNN. In the case of multi-task learning, T_s supervised tasks are considered to have a training dataset, $\mathcal{D}_i = (\mathbf{x}_j^i, y_j^i)_{j=1}^N$, where $i \in \{1, 2, \ldots, T_s\}$. Let $\mathcal{L}(\cdot)$ be the loss incurred during the training; the objective of multi-tasking learning is to minimize the following term:

$$\min \sum_{i=1}^{T_s} \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}(y_j^i, \phi(\mathbf{x}_j^i)) + \lambda ||\theta||^2. \tag{6}$$

Remarks. The existing literature on weight sharing [55, 56, 62, 69, 70] helps in reducing the storage and computation requirement of the DNN. However, the complexity of weight sharing also comes into the picture while representing the DNN using minimal weights. Thus, it could be beneficial to reduce the complexity of weight sharing in the DNN. This will also serve as a future research direction.

5 BITS PRECISION

This section presents an overview of the DNN compression technique that incorporates bits precision for reducing storage and computation requirements. In bits precision, the number of bits required for representing weight matrices is suppressed for reducing the storage and computation [115]. For example, we require 32 bits for storing the weights in the weight matrix in FLOPs of the DNN. It can be reduced to 8 bits by performing operations using integers. Table 4 summarizes different DNN compression techniques using bits precision.

5.1 Low Bits Representation

This section extends the concept of the integer representation to reduce the storage and computation of a DNN. It generalizes bits precision for using any number of bits for representing weights

Table 4. Summary of the DNN Compression Incorporating Bits Precision to Reduce Computation and Storage

Paper	Address the Challenge	Proposed Solution	Suitable for			Type
			CNN	FC	RNN	
[25]	Adopt integer arithmetic by replacing FLOPs	Co-processor architecture for toggling availability resources	✓	✓	✓	Low bits representation
[71]	To reduce the temporary storage for computation	Quantization mechanism that uses integers	✓	✓	✓	
[72]	To design a DNN that requires low-precision weights and activations	Linear and logarithmic scheme to reduce the bits requirements	✓	✓	✓	
[55]	Compress the DNN by reducing the bits required for matrices representation	Weight quantization to reduce the storage requirement of a DNN	✓	✓	✓	
[57]	To provide a better selection mechanism to deploy a DNN	Guided deployment for wiser pruning	✓	✓	✓	
[64]	Pruning method to deploy a DNN on embedded devices	Singular decomposition factorization method	✗	✗	✓	
[71]	Inference is performed using integer arithmetic	Quantization technique is an affine mapping	✓	✓	✓	
[73]	To replace FLOPs in a DNN with binary operations	Deterministic and stochastic techniques	✓	✓	✓	Binarization
[74]	Heterogeneous architecture to achieve higher flexibility	Parametric architecture for dataflow and optimization	✓	✓	✓	
[26]	Dividing the task into multiple subtasks and using binarization of weights	Quantitative measure to estimate the reduction in resources	✓	✗	✗	

instead of 8-bit integers only. The concept of **Quantized Neural Networks (QNNs)** is discussed by Hubara et al. [72]. The authors design a DNN network that requires very low precision (e.g., 1 bits, 4 bits, and so on) weights and activations during inference. While training, the quantized weights and activations help in estimating the gradient. The QNN scheme highly reduced the memory requirements and access latency, and replaced FLOPs with bit-wise operations. The authors used two types of quantization schemes to reduced the bits requirements for representing weights and activations: linear and logarithmic. The linear quantization $L_Q(\cdot)$ is defined as follows:

$$L_Q(r, l_b) = Clip\left(round\left(\frac{r}{2^{l_b}-1}\right)2^{l_b-1}, V_{\min}, V_{\max}\right), \tag{7}$$

where r is the real valued number, l_b is the length of bits, and $round(\cdot)$ is the round-off function. $V_{\min}$ and V_{max} are the minimum and maximum scale range, respectively. Further, the logarithmic quantization $Log_Q(\cdot)$ is given as follows:

$$Log_Q(r, l_b) = Clip\left(\log ap2(r), -(2^{l_b-1}-1), 2^{l_b-1}\right), \tag{8}$$

where $\log ap2(r)$ denotes the log of an approximate power of 2, and it uses the logarithm.

Zhou et al. [116] proposed a mechanism for training a CNN that used low bit weights and activations using low bit gradients. The authors stochastically quantized the gradient parameters during backpropagation using low bit numbers. It accelerated the training of the CNN, as forward and backward propagations are operated on low bit weights and activations. The authors named the approach *DoReFa-Net*, which used bit convolution kernels to accelerate inference. The authors

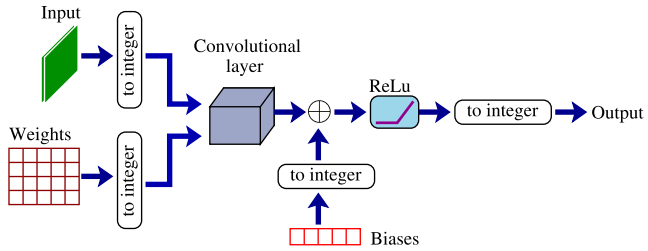

Fig. 5. Process of converting FLOPs into integer operations in a DNN.

claimed that the bit convolution in DoReFa-Net can be efficiently implemented on CPU, FPGA, and GPU. The bit convolution kernel in a low-bit DNN is defined as follows.

Let us consider that $\mathbf{x}$ is a sequence of a ω_1 fixed-point integer such that $\mathbf{x} = \sum_{i=0}^{\omega_1-1} c_i(\mathbf{x})2^i$. Further, assume that $\mathbf{y}$ is a fixed ω_2 bit integer such that $\mathbf{y} = \sum_{j=0}^{\omega_2-1} c_j(\mathbf{y})2^j$, where $\{c_i(\mathbf{x})\}_{i=0}^{\omega_1-1} \in \{0,1\}$ and $\{c_j(\mathbf{y})\}_{j=0}^{\omega_2-1} \in \{0,1\}$ are bit vectors. The dot product of $\mathbf{x}$ and $\mathbf{y}$ is estimated using the bit-wise operations given by the following expression:

$$\mathbf{x}.\mathbf{y} = \sum_{i=0}^{\omega_1-1} \sum_{j=0}^{\omega_2-1} 2^{i+j} bitcount[\&\&(c_i(\mathbf{x})c_j(\mathbf{y}))]. \tag{9}$$

Lee and Nirjon [25] proposed a mechanism of opportunistically accelerating the inference performance of the DNN, named *Neuro.ZERO*. They adopt four accelerating mechanisms: extended inference, expedited inference, ensemble inference, and latent training. Further, the authors proposed two algorithms for applying these accelerating mechanisms.

Jacob et al. [71] proposed a quantization mechanism that uses integer arithmetic despite FLOPs. It relies on the fact that multiplications are inefficient if the operands are wide (floating points 32 bits), and eliminating multiplication leads to performance degradation. Therefore, it could be beneficial to adopt integer multiplication despite FLOPs to reduce the temporary storage for computation. Figure 5 illustrates all of steps involved in performing integer operations by replacing all floating points.

Chauhan et al. [64] compared the performance of two systems—user authentication and user verification—deployed on RCDs. They incorporated weight quantization and factorization of FC layers to reduce the resource requirements of the DNN. Han et al. [55] proposed a weight quantization technique to compress the DNN by curtailing the number of bits required for representing the weight matrices. Here, the authors try to reduce the number of weights that should store in the memory. Let us consider a scenario where a deep learning model has four neurons at the input and output layers. Next, the weights are quantized to four bins, where all weights in the same bin share the same value. Thus, during the deployment of the DNN on the embedded device, we have to store smaller indices for each weight, as illustrated in Figure 6(a). Similarly, the gradients are also updated during the backpropagation of the DNN and weights are reduced, as illustrated in Figure 6(b).

Jacob et al. [71] proposed a quantization technique where the inference is performed using integer arithmetic. The integer arithmetic provides higher efficiency than FLOPs and requires a lower number of bits for representation. The authors used integer arithmetic during inference and FLOPs during training. The quantization technique is an affine mapping of integers Q to the real number R—that is, of the form $R = W(Q - T)$, where the quantization scheme has quantization parameters W and T. For example, for 8-bit quantization, Q is set to 8. The quantization parameter W is an arbitrary positive real number, and T is the same as Q.

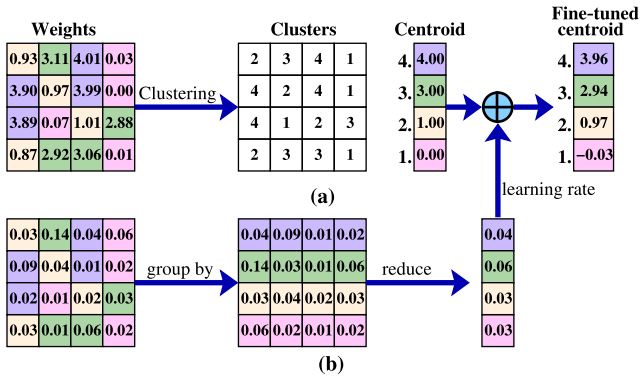

Fig. 6. Illustration of low bits representation [55]: Forward propagation (a) and backward propagation (b).

Remarks. The low bits representation discussed in prior studies [25, 55, 57, 64, 71, 71, 72] for compressing DNN has reduced the bits requirement for storage and execution. However, selecting an optimal number of bits for any estimation is tedious. Estimation using integers is an extension of low bits representation and provides the mechanism of using integers in place of floating-point values. This gives a future direction to develop a low bits representation mechanism that can optimally estimate the exact number of bits to achieve the desired accuracy.

5.2 Binarization

In this section, we present an overview of bits precision techniques [26, 73, 74] that incorporates binary numbers for performing different operations of the DNN. Binarization refers to the process of converting FLOPs into binary operations for reducing the storage and processing requirement of the DNN [117]. Yang et al. [26] presented a compact DNN architecture where a task is divided into multiple subtasks for reducing the resource requirement of the DNN. The authors named the approach *cDeepArch*. The decomposition of the task in cDeepArch efficiently utilizes the limited storage and processing capacity during the execution of the task. The authors emphasized that the unorganized compression of the DNN results in unreliable and low performance for recognizing multiple tasks.

Courbariaux et al. [73] introduced a training method that incorporates binary activations and weights. The resultant network is called the *Binary Neural Network* (BNN). Binarization helps in reducing storage requirements and provides energy efficiency. The authors highlighted two binarization techniques: deterministic and stochastic. In deterministic binarization, real-valued number r is transformed into binary value b using the $Sign(\cdot)$ function, which is given as follows:

$$b = Sign(r) = \begin{cases} +1 & \text{if } r \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{10}$$

In stochastic binarization, a probability p is estimated to convert real value r to binary value b as follows:

$$b = \begin{cases} +1 & \text{if } p = \max\left(0, \min\left(1, \frac{r+1}{2}\right)\right), \\ -1 & \text{if } 1 - p. \end{cases} \tag{11}$$

Umuroglu et al. [74] presented a framework for building a fast and flexible DNN compression mechanism, named *FINN*. It provides a heterogeneous architecture that achieves high-order flexibility. The framework is a parametric architecture for dataflow and an optimized method for

Table 5. Illustration of Knowledge Distillation Techniques for Improving the Performance of Compressed DNNs

Paper	Address the Challenge	Proposed Solution	Suitable for			Type
			CNN	FC	RNN	
[75]	Enhancing performance of single-layer DNNs	Knowledge distillation from LSTM using logits transfer	✗	✗	✓	Logits transfer
[77]	Improving performance of compressed DNNs	Knowledge distillation for improving performance	✓	✓	✓	
[78]	Distilling the distribution among same class labels	Performance using self-knowledge distillation	✓	✓	✓	
[76]	DNN compression provides significant privacy to the user	Design a private model compression framework	✓	✓	✓	
[79]	To highlight complexity for fine-tuning	Establish a relationship of compression and epochs	✓	✓	✓	
[80]	Interpretation of knowledge distillation	Involve relevant and irrelevant features of the visual task	✓	✓	✓	
[81]	To reduce the gap between student and teacher	Introduce an assistant between teacher and student	✓	✓	✓	Teaching assistant
[85]	To improve performance of a compact DNN	Simultaneous training of teacher and student	✓	✓	✓	
[86]	To reduce the variation in prediction probabilities	Incorporate layers sharing between teacher and student	✓	✓	✓	
[87]	For significantly improving performance of the student	Pre-trained and untrained teachers to guide the student	✓	✓	✓	
[82]	To train a few parameter students using a large teacher	Develop a robust technique for the open-set problem	✓	✗	✗	Domain distillation
[83]	Perform training on a simple DNN to handle domain shift	Learn transferable features for domain adaptation	✓	✗	✗	

classification on limited-resource devices. The authors implemented FC, convolutional, and pooling layers with computation as per the requirements of the user.

Remarks. Binarization of the DNN presented in the prior studies [26, 73, 74] reduces the storage and computations. The proposed binarization approaches achieve a high-order compression but with performance degradation. Thus, it would be beneficial, and serve as the future work, to mitigate such degradation for the successful adaptation of binarization in DNN compression.

6 KNOWLEDGE DISTILLATION

This section gives an overview of the DNN compression techniques that adopt knowledge distillation [118] to improve the performance of the compressed DNN. We consider knowledge distillation in this work in conjunction with the title, where the transformation generates lightweight DNNs from large-sized DNNs. Moreover, recent studies [119–121] have also stated knowledge distillation as a popular DNN compression technique, which supports our considerations and encourages us to summarize prior studies on knowledge distillation. The existing literature on knowledge distillation [75–83, 85–87] is further classified into three parts—logits transfer, teacher assistant, and domain adaptation—as illustrated in Table 5. In logits transfer, the unnormalized output of the original DNN (teacher) acts as a soft target for training the compressed DNN (student) [75, 122]. However, sometimes the gap between student and teacher is large, which hinders the knowledge transmission. This could be mitigated by inserting a teaching assistant in between the teacher and student.

6.1 Logits Transfer

Logits transfer is the simplified approach for knowledge distillation from the teacher to student. First, the teacher is trained on a given dataset $\mathcal{D}$. Next, the logit vectors (output of the DNN before the softmax layer) of the teacher acts as a soft target for a training student. The knowledge distillation using logits transfer improved the generalization ability of the student and helped in improving performance. Chen et al. [75] proposed a knowledge distillation technique for recognizing human activities using a single-layer feed-forward neural network to extract statistical features under the guidance of a cumbersome DNN (dense LSTM network).

Hinton et al. [77] proposed a knowledge distillation technique for improving the performance of a compressed DNN using the generalization ability of the pre-trained cumbersome DNN. In the DNN, the output features vector obtained at one layer prior to the softmax layer is termed *logit*. Let $\mathbf{a}_i$ denote a logit vector of data instance i ($\forall i \in N$), where N represents total number of data instances in dataset $\mathcal{D}$. To obtain $\mathbf{a}_i$, let $x_{ij} \in \mathbf{X}$, $w_{ij} \in \mathbf{W}^T$, and $b_j \in \mathbf{b}$ represent an element of feature matrix, weight matrix, and bias vector for the j^{th} ($1 \leq j \leq l$) class of the i^{th} ($1 \leq i \leq N$) training instance, respectively, where l denotes the number of classes in $\mathcal{D}$. Hence, we can estimate an element a_{ij} of logit vector $\mathbf{a}_i$ for class j given as $a_{ij} = w_{ij}x_{ij} + b_j$. The logits vector $\mathbf{a}_i = \{a_{ij} | 1 \leq j \leq l\}$ passes to a softmax function to compute predicted class label probability $p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{l} e^{a_{ij}}}$.

The predicted probability vector for the i^{th} instance is a set of probabilities $\{p_{i1}, p_{i2}, \ldots, p_{il}\}$ against each class label l and a class label with the highest probability value is said to be the predicted class label. Further, we introduce a variable $\mathcal{T}$ for generating a softer probability distribution: $\rho_{ij} = \frac{e^{u_{ij}/\mathcal{T}}}{\sum_{j=1}^{k} e^{u_{ij}/\mathcal{T}}}$.

To perform the matching of logits, we have to estimate gradient ∇ with the element (a_{ij}) of logit vector. a_{ij} is the element of logit vector of the i^{th} data instance and the j^{th} class. Let b_{ij} represent the element of logit vector of the cumbersome DNN, then the gradient ∇ is estimated as follows:

$$\nabla = \sum_{j=1}^{l} \frac{1}{\mathcal{T}} \left(\frac{e^{a_{ij}/\mathcal{T}}}{\sum_{j=1}^{l} e^{a_{ij}/\mathcal{T}}} - \frac{e^{b_{ij}/\mathcal{T}}}{\sum_{j=1}^{l} e^{b_{ij}/\mathcal{T}}} \right). \tag{12}$$

At the higher value of $\mathcal{T}$, the magnitude of matching logits in Equation (12) can be rewritten in an approximate format as follows:

$$\Delta \approx \sum_{j=1}^{l} \frac{1}{\mathcal{T}} \left(\frac{1 + a_{ij}/\mathcal{T}}{N + \sum_{j=1}^{l} a_{ij}/\mathcal{T}} - \frac{1 + b_{ij}/\mathcal{T}}{N + \sum_{j=1}^{l} b_{ij}/\mathcal{T}} \right). \tag{13}$$

The main objective of knowledge distillation is to minimize the gradient (∇) in Equation (13). $\nabla = 0$ indicates the performance of the cumbersome and compressed DNN is precisely the same. However, it is impractical to reach this equilibrium state, as it is achieved when the compressed DNN is trained for an infinite number of epochs with logits of the teacher model. To avoid such a problem, we stop training the compressed DNN when $\nabla \to 0$. Figure 7 illustrates the steps involved in the knowledge distillation using logits transfer.

Yun et al. [78] distilled the predictive distribution among same class labels for training. The distillation results in the regularization of knowledge obtained from wrong predictions, in a single DNN. The authors combined two loss functions—cross entropy and KL divergence—using a regularization parameter λ. The combined loss function $\mathcal{L}_{comb}(\cdot)$ is given as follows:

$$\mathcal{L}_{comb}(\mathbf{x}, \mathbf{x}', y, \phi, \mathcal{T}) = \mathcal{L}_{CE}(\mathbf{x}, y, \phi) + \lambda \mathcal{T}^2 \mathcal{L}_{KL}(\mathbf{x}, \mathbf{x}', \phi, \mathcal{T}),$$

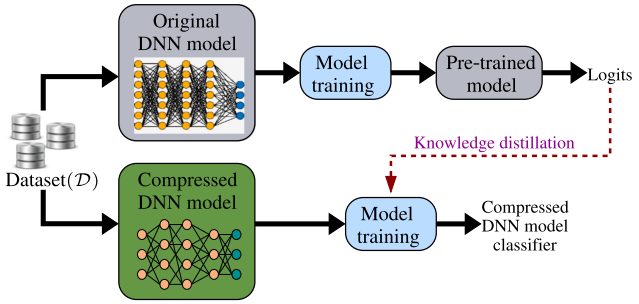

Fig. 7. Illustration of knowledge distillation using logits.

where $\mathcal{L}_{CE}(\cdot)$ and $\mathcal{L}_{KL}(\cdot)$ are the cross-entropy and KL divergence losses, respectively. $\mathbf{x}$ is the data instance instance having label y and $\mathbf{x}'$ is another data instance having label y. ϕ is the network parameter and $\mathcal{T}$ is the temperature used in the knowledge distillation for softening the prediction probabilities of the knowledge inferring model.

Wang et al. [76] designed a private model compression framework, named *pRivate mOdel compressioN frAmework* (RONA). The authors highlighted the need for DNN compression to provide significant privacy to the user by analyzing the collected data on the limited-resource devices. Using knowledge distillation, the accuracy of the compressed model is improved by jointly using a hint, distillation, and self-learnings. During knowledge distillation, the cumbersome model is carefully perturbed to enforce a high level of privacy. Li et al. [79] highlighted the complexity of fine-tuning of a compressed DNN. The compression of DNN using pruning, quantization, and weight decomposition requires fine-tuning, which may take many epochs for stabilization. Another problem associated with fine-tuning is the compressed DNN requiring a large training dataset to achieve considerable accuracy. Here, the authors try to realize both data and processing efficiency.

Cheng et al. [80] presented a method for successful interpretation of knowledge distillation that involves both relevant and irrelevant features related to the visual task. They presented three types of hypotheses for knowledge distillation: (1) it helps in learning more precise class label predictions from raw data, (2) it ensures simultaneously learning multiple classes from the dataset, and (3) it provides the well-optimized direction for learning. The authors have demonstrated the pros and cons of the hypothesis.

Remarks. The existing work on logits transfer [75–80] in knowledge distillation improves the performance of the compressed DNN. However, the logits of the original pre-trained DNN sometimes lead to overfitting, which leads to poor generalization ability of the compressed model. During fine-tuning of a compressed DNN, the student can mimic the exact prediction behavior as the teacher but lose its generalization ability. It could be a future research direction to determine the exact point for halting the training of the student to retain its generalization ability.

6.2 Teaching Assistant

Several authors [81, 85–87] studied the concept of knowledge distillation from a different perspective. The authors introduced teaching assistants in between the teacher and student during knowledge distillation. This insertion of teaching assistants has the main motive to reduce the gap between the student and teacher. They claimed that when the student's size is fixed, we cannot employ a prominent teacher. To mitigate such difficulty in improving the performance of the student through knowledge distillation, the teaching assistant plays a decisive role.

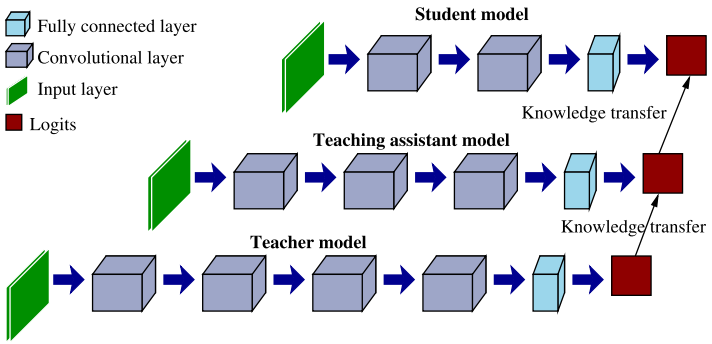

Fig. 8. An example scenario of a teaching assistant between a teacher and student [81].

Figure 8 illustrates an example scenario that demonstrates the role of a teaching assistant to fill the gap between a large teacher and a small student. Mirzadeh et al. [81] suggested the futuristic approach of inserting multiple teaching assistants for providing high-order compression with minimal accuracy compromise. Zhou et al. [86] adopted the concept of layer sharing, where a few initial layers of the student and the untrained teacher are shared to reduce the training time and improve the softmax (prediction) probabilities. Here, the authors followed the mechanism that resembled the rocket launching mechanism. Different from earlier approaches, Zhao et al. [87] used the concept of multiple teachers, where the training of the student is guided by a pre-trained teacher and an untrained teacher. The authors claimed to achieve significant improvement in accuracy due to multiple teachers.

Remarks. Mirzadeh et al. [81] discussed the benefits of inserting a teaching assistant between the teacher and student, but unfortunately it will increase the training time by nearly twofold. First, the training of teaching assistants is performed using a teacher followed by training of a student using a teaching assistant. Despite significant improvement in student performance, the complexity of training could be a loophole of the prior work that can be a future research direction.

6.3 Domain Adaptation

Knowledge distillation techniques discussed in the previous sections do not consider the consequence of domain disparity while training a student using the logits of the teacher. The domain disparity leads to poor generalization ability of the student [123]. To handle domain disparity, different domain adaptation techniques have been proposed in the existing literature [76, 82, 83]. Figure 9 illustrates the layer-by-layer feature extraction from the teacher and student followed by knowledge distillation from the teacher to student.

Duong et al. [82] proposed a knowledge distillation technique that helped in shrinking the gap between a cumbersome teacher and a compact student. Therefore, the authors called this technique *ShrinkTeaNet*. The objective of ShrinkTeaNet is simple to train a few parameter students using a cumbersome teacher. Further, the authors introduced angular distillation loss for determining the feature direction despite the hard targets that result in overfitting. The angular distillation loss ($\mathcal{L}_{adl}(S,T)$) is defined as follows:

$$\mathcal{L}_{adl}(S,T) = dist\Big(trans(F_t), trans(F_s)\Big) = \left\| 1 - \frac{trans(F_t)}{trans(F_t)} \times \frac{trans(F_s)}{trans(F_s)} \right\|_2^2, \tag{14}$$

where $dist(\cdot)$ is the distance function that estimates the distance between the transformation function of the teacher ($trans(F_t)$) and student ($trans(F_s)$). Equation (14) is similar to cosine similarity for estimating the distance between vectors.

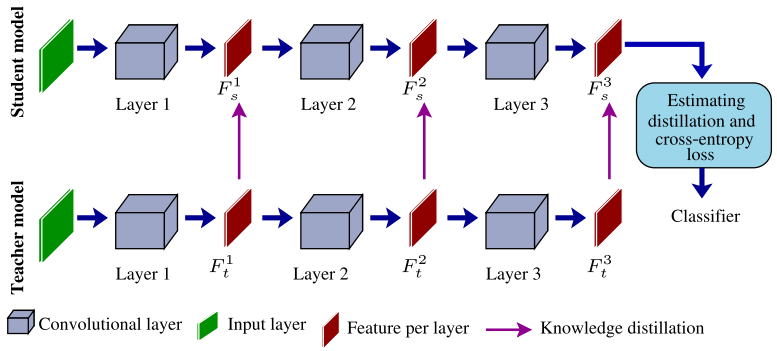

Fig. 9. Layer-by-layer knowledge distillation from teacher to student for eliminating domain disparity.

Table 6. Illustration of Impact of Knowledge Distillation on the Performance of Lightweight DNNs

DNN	Achieved accuracy on a		
	Large-Sized DNN	Lightweight DNN	Distilled Lightweight DNN
DeepZero [6]	93.44%	85.19%	92.21%
DeepFusion [30]	94.64%	83.03%	91.83%
DeepSense [23]	93.01%	80.20%	89.19%

Yang et al. [83] emphasized the problem of domain adaptation encountered during the training of the DNN on embedded IoT devices. The authors discovered that the DNN achieves superior results on the high-end machine but suffers from degradation in accuracy and efficiency on IoT devices. They proposed a framework for Mobile Domain Adaptation (MobileDA) that can learn the transferable features. The learning is performed in such a way that the structure of the DNN remains as simplified as possible.

Remarks. The domain adaptation technique proposed in the existing literature [76, 82, 83] has successfully covered the domain disparity problem. However, the gap between the student and teacher sometimes hampers the performance of the domain adaptation technique. Thus, it could be beneficial to adopt the teaching assistant in the domain adaptation problem to get overall benefits. This would be a prominent research direction.

Experimental Outcome. To determine the experimental outcome, we consider the same large-sized DNN used in the Experimental Outcome of Section 3 (i.e., DeepZero [6], DeepFusion [30], and DeepSense [23]). We apply knowledge distillation to improve the performance of the lightweight DNN (40% compressed) using the large-sized DNN in Table 2 on the SHL dataset [124]. The results in Table 6 illustrate the impact of knowledge distillation on the performance of the lightweight DNN. We observed a significant increase in accuracy by adopting knowledge distillation. Such improvement depends on the selected lightweight and large-sized DNN and the considered dataset. We used the baseline technique during the experiment as discussed by Hinton et al. [77].

7 MISCELLANEOUS

This section covers the compression techniques in the literature [19–24, 27–30, 88–94] that do not meet the different categorization criteria discussed in the previous sections. Apart from the outlier of categorization criteria, some of them simultaneously incorporate a group of compression

Table 7. Summary of Miscellaneous Approaches for Compressing DNNs

Paper	Address the Challenge	Proposed Solution	Suitable for		
			CNN	FC	RNN
[19]	Recognize human activities via RCDs from sensory data	Select a compressed and device-specific DNN using resources at RCDs	✓	✓	✓
[20]	Reduce memory, energy, and computation without loss	Technique to decompose the DNN into multiple unit blocks	✓	✓	✓
[21]	To determine a device-specific DNN to recognize activities	Exploit convolutional and recurrent units on a sensory dataset	✓	✓	✓
[22]	Compression to enhance suitability for a smartphone	Develop a smartphone-specific framework to compress RNNs	✗	✗	✓
[23]	Handle noise in the dataset collected from mobiles	Unified framework for on-device classification	✓	✓	✓
[24]	Exploit non-linear relation of the DNN and execution time	Determine the tradeoff between execution time and accuracy	✓	✓	✓
[88]	Automatically specify compression techniques	Determine an optimal balance of resources and user demand	✓	✓	✓
[27]	To determine capability of the DNN to perform recognition	To utilize cross-sensor correlation, providing higher performance	✓	✗	✗
[89]	Emphasize the dynamic resources of mobile devices	Select optimal resources to perform inference on the mobile device	✓	✓	✓
[28]	Using multimodal sensors to recognize embedded devices	Using the modality-specific partition	✓	✓	✗
[90]	To perform recognition with high accuracy in a short time	Optimizing operations of convolutional and FC layers	✓	✓	✗
[29]	Determine the device usage pattern using historical data	Deep reinforcement learning framework	✓	✓	✓
[30]	Recognition of different IoT-based tasks	Framework that incorporates values from multiple sensors	✓	✓	✓
[92]	To reduce the resource via distributed operations	Implement a technique to train the DNN in a distributed manner	✓	✓	✓
[93]	Issues associated with the DNN on high-end machines and RCDs	Elaborate the contradiction among requirements of the DNN and devices	✓	✓	✓
[94]	Reduce power to minimize FLOPs without accuracy	Use pointwise group convolution and channel shuffle	✓	✗	✗

techniques. An overview of compression techniques under the miscellaneous category is illustrated in Table 7.

Jeyakumar et al. [19] proposed an optimal compressed selection mechanism named *SenseHAR* for recognizing human activities using sensory values. The authors used the device's information in determining the model to be deployed on the device. Lane et al. [20] presented a software-based DNN accelerator to provide inference on RCDs. The authors named the technique *DeepX*, which can significantly reduce memory, energy, and processing requirements of the DNN. Hammerla et al. [21] exploited DNN architectures, including convolutional and recurrent units on the sensory dataset, and tried to determine the most appropriate model for a device. They studied the suitability of a DNN for the multiple sensory datasets on human activities recognition. Cao et al. [22] proposed a DNN compression technique named *MobiRNN* for compressing an RNN. This compression enabled the deployment of a computationally intensive RNN on the smartphone.

Radu et al. [27] mainly focused on determining the capability of a DNN to perform recognition of sensory values on mobile and embedded devices. The authors utilized cross-sensor correlation for providing a high order of recognition performance. Fang et al. [89] presented a DNN framework named *NestDNN*. The framework emphasized the availability of dynamic resources. Yao et al. [23] proposed a unified framework that can handle the noise insertion in the collected dataset due to on-device sensors. Along with noise handling, the on-device classification is also beneficial to reduce the inherent delay due to data transmission from the device to the high-end machine. Yao et al. [24] proposed a framework that provided an insight into the black box of a DNN. The framework is named *FastDeepIoT*, which provides an accuracy preserving DNN compression for IoT devices.

Liu et al. [88] explored the tradeoff between performance and resources of the DNN on the embedded device. Here, the authors determined the user-specified needs to estimate the appropriate DNN for its deployment on the embedded device. The authors refer to this framework as *AdaDeep*. The framework automatically specifies a combination of compression techniques for a given DNN. AdaDeep tried to solve the following optimization problem:

$$\arg\max_{c_t \in C_t} \lambda_1 \tau(acc - acc_{min}) + \lambda_2 \tau(E_{max} - E), \text{ subject to, } L \leq L_{th}, \text{ and } St \leq St_{th},$$

where accuracy (acc), energy (E), latency (L), and storage (St) have the corresponding threshold acc_{min}, E_{max}, L_{th}, and St_{th}, respectively. The accuracy and energy constants are combined using constraints λ_1 and λ_2. $\tau(\cdot)$ is the normalization process for both accuracy and energy.

Radu et al. [28] studied the different benefits of using multimodal sensors for recognizing various activities using a DNN classifier on embedded devices. They mainly emphasized two variants of DNN—the FC layer and CNNs—that are using the concatenation of multi-sensor values. Here, the authors used the modality-specific partition. Mathur et al. [90] presented a DNN compression technique called *DeepEye*.

Xu et al. [91] proposed a cache designing mechanism that helps in providing inference on mobile devices. The authors named the approach *DeepCache*. They suggested that cache memory be available and handle the memory variation in the raw sensory data. The caching provides space to store results and helps in exploiting result reusability. Shen et al. [29] proposed a deep reinforcement learning framework named *DeepApp*. It determined the mobile device usage pattern using historical data of the applications being used by the user. The features from the historical data are extracted using a compressed DNN. The compressed DNN classifier recognized the different usage patterns.

Xue et al. [30] proposed a unified framework (named *DeepFusion*) that incorporated values of multiple sensors for recognition of different IoT-based tasks. DeepFusion parallelly combines spatial features extracted from the CNN for each sensory value. The multimodal sensory value is considered, as it provides better distinguishable characteristics and reduces the noise in the sensory data.

Wang et al. [93] elaborated on the contradiction between resource requirements of the DNN and deployment of the DNN on miniature mobile devices having limited resources. Other issues associated with training of the DNN on the high-end machine is the privacy and security concern about individual data. Zhang et al. [94] incorporated two crucial operations—pointwise group convolution and channel shuffle—for compressing a DNN. The proposed approach was named *ShuffleNet*, which reduces the power consumption by lowering the FLOPs.

Sandler et al. [125] described a new mobile architecture, named *MobileNetV2*, to shrink the DNN for IoT applications. The architecture helped in improving the performance of the mobile models for multi-tasking applications across different sizes. The authors employed the proposed MobileNetV2 for the task of object detection. The architecture used the inverted residual structure

with shortcut connections among narrow bottleneck layers. The authors obtained that it is crucial to eliminate non-linearities among narrow layers; thus, the obtained module is an inverted residual with the linear bottleneck. The module has taken the low-dimension compressed input, which is initially expanded to a high dimension.

Depthwise separable convolution replaced the full convolutional operation with a factorized version. During factorization, the convolutional layers are split into two separate layers. The first layer, called *depthwise convolution*, performs filtering using single convolutional filtering per channel. The second layer performs 1×1 convolution (pointwise convolution). The pointwise convolution built new features by linearly combining the input channels. Let $k \times k \times d_c \times d_o$ denote the dimension of the kernel applied on the convolutional layer having an input tensor, denoted as l_c, of dimension $h_c \times w_c \times d_c$. The application of a convolutional kernel, denoted as $K \in \mathbb{R}^{k \times k \times d_c \times d_o}$, on l_c produced output tensor l_o of dimension $h_c \times w_c \times d_o$. The computational requirement of the standard convolutional layer is given by the following expression: $h_c \times w_c \times d_c \times d_o \times k \times k$. The cost of depthwise and pointwise convolution can be obtained from Sandler et al. [125], as follows: $h_c \times w_c \times d_c(k^2 + d_o)$.

Ma et al. [126] highlighted that the design of DNN architectures is guided mainly by FLOPs (i.e.,, an indirect metric). Therefore, the authors suggested using direct metrics such as speed, which depends on factors like memory cost and platform properties. The discrepancy between FLOPs and speed is mainly due to two reasons. First, FLOPs do not consider factors such as memory cost, platform properties, and degree of parallelism. Second, the operation with the same FLOPs may execute at different times, varying from one machine to another. Further, according to the guidance of direct metrics, the authors proposed a new architecture, named *ShuffleNetV2*. The architecture is suitable for low-end devices like smartphones. ShuffleNetV2 uses channel split, where at the beginning of each unit, the input of f feature channels are split into two branches, $f1$ and $f2$ channels, at given $f = f1 + f2$. One branch is set to be constant, and the other comprises three convolutions having the same input and output.

Remarks. The DNN compression techniques SenseHAR [19], DeepX [20], MobiRNN [22], DeepSense [23], FastDeepIoT [24], AdaDeep [88], NestDNN [89], DeepEye [90], DeepCache [91], DeepApp [29], DeepFusion [30], ShuffleNet [94], and others [21, 28, 92, 93] in the miscellaneous approach have reduced the storage and computation requirements with a different perspective.

8 IOT APPLICATIONS USING A COMPRESSED DNN

This section discusses different IoT applications using a compressed DNN. It first categorizes and then discusses low-level and high-level applications of IoT using a compressed DNN. The low-level applications of IoT demand a compressed and trained lightweight DNN suitable for low-cost, low-powered, and compact RCDs. The low-level applications perform inference from the deployed model. Contrarily, the high-level applications demand a lightweight DNN for various compact devices with heterogeneous resources. Further, the low-level applications are limited to individual use, and high-level applications impact a large-sized population. The low-level application includes indoor localization [127–129], image and video processing [130–133], text and voice recognition [134, 135], monitoring and control [133, 136, 137], and security and privacy [138, 139].

Indoor Localization. GPS-based navigation heavily relies on satellite signals; thus, they do not work for localization in an indoor environment. Indoor localization is one of the potential applications of IoT, where radio signals are processed and analyzed to locate objects in indoor environments. Such radio signal can be obtained from WiFi and Bluetooth of compact IoT devices like smartphones and Raspberry Pi, among others. However, it is challenging to perform computation for indoor localization on IoT devices [127–129]. Tiku et al. [127] presented an approach to curtail the computational complexity of DNN-based indoor localization. The authors claimed to maintain

localization accuracy despite the compact size of DNN and reduced the response time. The proposed approach optimized CNN for indoor localization and facilitated its deployment on mobile devices. The compact CNN predicted the indoor coordinates based on the previous values with high precision.

Image and Video Processing. IoT-based applications generate a large amount of image and video datasets. The growth in such datasets is accelerated by the technological advancement in image sensing. It facilitates real-time monitoring and control in surveillance, face recognition, and auto verification of the product quality. To process the image or video dataset, it is required to use a large-sized DNN that cannot be executed on compact IoT devices [130–133]. Tian et al. [130] highlighted the problem of human-made criteria to prune filters—that is, it requires complicated knowledge of prior information. To address the problem, the authors proposed an efficient pruning framework for jointly pruning and learning a lightweight DNN simultaneously. The framework first adds a binary collaborative layer to each existing layer. Next, an asymptotic gradient estimator is passed in the binary collaborative layer. Finally, sparse structure and optimal weights are simultaneously learned from the large-sized DNN. The framework is tested against existing image datasets, where the framework proved its superiorities.

Text and Audio Processing. Audio and text processing advancements proliferated many applications in IoT. In addition, human-computer interaction also introduced the need for audio and text applications. Compact devices like smartphones are the primary source of audio or text interaction [134, 135]. Hu et al. [134] proposed a domain robust DNN-based mechanism for verifying speakers. The authors introduced a multi-task framework to determine the speakers' embeddings in an end-to-end manner. The framework can work with labeled and unlabeled datasets.

Monitoring and Control. Monitoring and controlling via compact IoT devices are more prominent applications that facilitate various domains, including agriculture, education, healthcare, and transportation. DNNs are popular among monitoring and control applications and well process such tasks [133, 136, 137]. Hassantabar et al. [136] highlighted the challenge of continuously monitoring human health using wearable medical sensors and a compact DNN's deployment on edge devices. The authors proposed a framework, called *MHDeep*, which incorporates wearable medical sensors and a lightweight DNN for diagnosing critical medical issues such as schizoaffective, major depressive, and bipolar disorders. They utilized eight categories of data from various sensors embedded in smartwatches and smartphones. MHDeep used the DNN, which vanishes extracting features manually.

Security and Privacy. The growing advancements in DNN compression also raise security and privacy issues. Therefore, researchers have developed different mechanisms to ensure the privacy and security of the compact or lightweight DNN deployed on RCDs [138, 139]. Phan et al. [138] highlighted the security issues in the compressed DNN, especially backdoor attacks.

We discuss three important high-level applications, including the intelligent transportation system [6, 37, 95], river water monitoring [140–142], and UAV-enabled applications [143–146].

Intelligent tranportation system: The proliferation of compact and low-powered devices in IoT has affected intelligent transportation systems in different aspects, including road health monitoring [95], locomotion mode recognition [6], and driver assistance systems [37]. Mishra et al. [95] analyzed the available resources of the smartphone, including processing power, available memory, desired accuracy, and residual energy, to determine an optimal lightweight DNN for road health monitoring. Similarly, Mishra et al. [6] proposed a lightweight DNN suitable for the smartphone to recognize locomotion modes. The authors incorporated zero-shot learning to identify both seen and unseen locomotion modes, achieving higher accuracy. Gupta et al. [37] proposed an early classification approach that analyzed road health and quickly provided information to the driver for appropriate actions.

River Water Monitoring: Water quality assessment is one of the potential applications of IoT and provides an easier and convenient mechanism for real-time monitoring of water bodies like rivers [140–142]. Chopade et al. [140] proposed a sensor-based river water quality assessment system using a compact DNN while assuming limited data samples, unlabeled sensory data, and noise data. The authors built a compact DNN to predict river water quality against unknown instances.

UAV-Enabled Applications: IoT helps in accelerating UAV-enabled applications in different domains ranging from agriculture, healthcare, product delivery, surveillance, and localization [143–146]. Singh et al. [143] highlighted the importance of the Internet of Drones or UAVs in agriculture. They suggested that optimally using UAVs in agriculture results in a qualitative and quantitative improvement in production. The authors summarized different challenges and solutions encountered while using UAVs in agriculture. Such challenges include limited onboard resources, flying permission, accuracy, and delay, among others.

9 DISCUSSION AND FUTURE RESEARCH DIRECTIONS

IoT-based applications are rapidly growing day by day to induce a high level of automation and provide ease of real-time monitoring. The availability of low-cost, low-power, and small-sized sensors and microcontroller units accelerate this expeditious growth. IoT-based applications have influenced different domains, including healthcare, transportation, smart metering, environment monitoring, and agriculture, among others. Further, the significant amount of data generated in the IoT can be utilized for analysis and prediction of current situations and future events. Incorporating the DNN in these analyses and predictions can significantly improve their performance, which is requisite for critical applications like healthcare, traffic monitoring, and intruder detection in battlefields, among others. However, the colossal resource requirements of the DNN hamper its inclusion in delay-sensitive applications, which demands in situ processing. The presented categorization for compressing DNNs provided a glimpse of the existing literature and its colossal contributions. After a thorough review of prior studies on DNN compression for IoT, we came up with five broad categories: network pruning, sparse representation, bits precision, knowledge distillation, and miscellaneous techniques. All of these categories seem to be the predominating area of research that encourages effective utilization of the DNN on RCDs. The proliferating demand for IoT-based applications has compelled data scientists to incorporate the DNN in IoT. Therefore, the presented survey will become a milestone to develop new methodologies and approaches to meet the limited storage and processing capacity available at the resource-constrained IoT devices. The overview of different categories of DNN compression provides various research possibilities. Even though several studies have been done in DNN compression, there are still many more to be uncovered. Table 8 describes the list of symbols used in this article.

In summary, the future directions of transforming large-sized DNNs into lightweight ones can be categorized as follows: (1) reducing the training time of the compressed DNN, (2) automating the DNN compression, (3) utilizing a lightweight DNN in federated learning, (4) deployment of the lightweight DNN in IoT, and (5) exploiting a lightweight DNN in UAV applications. The time complexity of obtaining appropriate or resource-specific lightweight DNNs from a large-sized DNNs is tedious, as it requires iterative construction and training. These iterative steps are continued for many rounds, which is time consuming. Thus, *reducing the time to obtain a lightweight DNN* is a potential future direction. The available processing power and memory on IoT devices dynamically change; therefore, reducing the time helps in getting resource-varying lightweight DNNs suitable for time-critical applications like intruder detection, flood detection, forest fire detection, industrial control, robotics, smart driving, and smart home appliances. Manually determining the pruning layers, filters, neurons, and bits of representation is a complex task and may never reach

Table 8. List of Symbols Used in This Article

Symbol	Description	Symbol	Description	Symbol	Description
$\mathcal{D}$	Dataset	N	Instances in $\mathcal{D}$	k	Input dimension
M	Length of instance	$\mathcal{L}(\cdot)$	Loss function	acc	Accuracy
l	Classes in l	$Clip(\cdot)$	Clip function	Δ	Scaling factor
$Sign(\cdot)$	Sign function	$\mathcal{L}_{adl}(\cdot)$	Angular loss	E_{max}	Energy threshold
$trans(\cdot)$	Transformation	$\mathbf{X}$	Input matrix	D_{KL}	KL divergence
δ	Vector of length k	wd	Width multiplier	$\mathcal{S}_g$	Storage gain
rm	Resolution multiplier	c_{rate}	Compression rate	$\mathcal{C}_g$	Computation gain
y	Data label	E	Energy	Ω	Fixed mask
St	Storage requirement	L	Latency	Loss_c	Contrastive loss
$dist(\cdot,\cdot)$	Cosine distance	T_s	Supervised task	$L_Q(\cdot)$	Quantization
$round(\cdot)$	Rounding function	V_{min}	Minimum range	V_{max}	Maximum range
$\log ap$	Approx. parameter	ω_1	Fixed integer	St_{th}	Storage threshold
$Sign()$	Sign function	p	Probability	∇	Gradient
$\mathcal{T}$	Temperature of KD	ρ	Prob. distribution	$\mathcal{L}_{th}$	Loss threshold

an optimal value in finite time. Therefore, it creates a futuristic challenge of *automating DNN compression*. It can accelerates the IoT applications like consumer-level home automation, autonomous infotainment delivery, and machine-operated smart supply chain management systems. Federated learning is an emerging paradigm of privacy-preserving collaboration using multiple clients. Such clients possess heterogeneous resources; thus, we require a different configuration of lightweight DNNs for various clients. This flourishes a future direction of working on *DNN compression in federated learning*. DNN compression in federated learning provides the opportunity to handle heterogeneous participants. It helps toward the development of privacy-preserving and distributed applications for localization service, smart transportation, mobile packet classification, smart finance, object detection, and energy prediction. A more convenient future direction is to *deploy a lightweight DNN* in the real-world scenario of IoT. However, the mechanism of effectively training a deployed DNN on IoT devices is a challenging problem. It is suitable for applications like human activities recognition and security enhancement using wearables and compact devices, respectively. UAV-based applications dominate the modern era, impacting monitoring, surveillance, product delivery, and controlling operations.

REFERENCES

[1] D. Singh, G. Tripathi, and A. J. Jara. 2014. A survey of Internet-of-Things: Future vision, architecture, challenges and services. In *Proceedings of WF-IoT*. 287–292.

[2] Fatima Hussain, Rasheed Hussain, Syed Ali Hassan, and Ekram Hossain. 2020. Machine learning in IoT security: Current solutions and future challenges. *IEEE Communications Surveys & Tutorials* 22, 3 (2020), 1686–1721.

[3] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. 2018. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence* 2, 1 (2018), 41–50.

[4] Junfeng Xie, F. Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. 2018. A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 393–430.

[5] Shuochao Yao, Yiran Zhao, Aston Zhang, Shaohan Hu, Huajie Shao, Chao Zhang, Lu Su, and Tarek Abdelzaher. 2018. Deep learning for the Internet of Things. *Computer* 51, 5 (2018), 32–41.

[6] R. Mishra, A. Gupta, H. P. Gupta, and T. Dutta. 2022. A sensors based deep learning model for unseen locomotion mode identification using multiple semantic matrices. *IEEE Transactions on Mobile Computing* 21, 3 (2022), 799–810.

[7] P. Kumari, H. P. Gupta, and T. Dutta. 2020. An incentive mechanism-based Stackelberg game for scheduling of LoRa spreading factors. *IEEE Transactions on Network and Service Management* 17, 4 (2020), 2598–2609.

[8] A. Gupta, H. P. Gupta, B. Biswas, and T. Dutta. 2021. An unseen fault classification approach for smart appliances using ongoing multivariate time series. *IEEE Transactions on Industrial Informatics* 17, 6 (2021), 3731–3738.

[9] P. Kumari, H. Gupta, and T. Dutta. 2020. Estimation of time duration for using the allocated LoRa spreading factor: A game-theory approach. *IEEE Transactions on Vehicular Technology* 69, 10 (2020), 11090–11098.
[10] Weiping Ding, Yurui Ming, Zehong Cao, and Chin-Teng Lin. 2022. A generalized deep neural network approach for digital watermarking analysis. *IEEE Transactions on Emerging Topics in Computational Intelligence* 6, 3 (2022), 613–627.
[11] Ya Tu and Yun Lin. 2019. Deep neural network compression technique towards efficient digital signal modulation recognition in edge device. *IEEE Access* 7 (2019), 58113–58119.
[12] Ruthvik Vaila, John Chiasson, and Vishal Saxena. 2022. A deep unsupervised feature learning spiking neural network with binarized classification layers for the EMNIST classification. *IEEE Transactions on Emerging Topics in Computational Intelligence* 6, 1 (2022), 124–135.
[13] Yanan Liu, Xianbin Wang, Gary Boudreau, Akram Bin Sediq, and Hatem Abou-Zeid. 2020. Deep learning based hotspot prediction and beam management for adaptive virtual small cell in 5G networks. *IEEE Transactions on Emerging Topics in Computational Intelligence* 4, 1 (2020), 83–94.
[14] Ayten Ozge Akmandor, Hongxu Yin, and Niraj K. Jha. 2018. Smart, secure, yet energy-efficient, Internet-of-Things sensors. *IEEE Transactions on Multi-Scale Computing Systems* 4, 4 (2018), 914–930.
[15] Lin Wang and Andrea Cavallaro. 2021. Deep learning assisted time-frequency processing for speech enhancement on drones. *IEEE Transactions on Emerging Topics in Computational Intelligence* 5, 6 (2021), 871–881.
[16] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. 2017. DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of SenSys*. 1–14.
[17] Xiao Liu, Wenbin Li, Jing Huo, Lili Yao, and Yang Gao. 2020. Layerwise sparse coding for pruned deep neural networks with extreme compression ratio. In *Proceedings of AAAI*. 4900–4907.
[18] Seulki Lee and Shahriar Nirjon. 2020. Fast and scalable in-memory deep multitask learning via neural weight virtualization. In *Proceedings of MobiSys*. 175–190.
[19] Jeya Vikranth Jeyakumar, Liangzhen Lai, Naveen Suda, and Mani Srivastava. 2019. SenseHAR: A robust virtual activity sensor for smartphones and wearables. In *Proceedings of SenSys*. 15–28.
[20] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of IPSN*. 1–12.
[21] Nils Y. Hammerla, Shane Halloran, and Thomas Plötz. 2016. Deep, convolutional, and recurrent models for human activity recognition using wearables. In *Proceedings of IJCAI*. 1533–1540.
[22] Qingqing Cao, Niranjan Balasubramanian, and Aruna Balasubramanian. 2017. MobiRNN: Efficient recurrent neural network execution on mobile GPU. In *Proceedings of MobiSys*. 1–6.
[23] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. 2017. DeepSense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of WWW*. 351–360.
[24] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. 2018. FastDeepIoT: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of SenSys*. 278–291.
[25] Seulki Lee and Shahriar Nirjon. 2019. Neuro.ZERO: A zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of SenSys*. 138–152.
[26] Kang Yang, Tianzhang Xing, Yang Liu, Zhenjiang Li, Xiaoqing Gong, Xiaojiang Chen, and Dingyi Fang. 2019. cDeepArch: A compact deep neural network architecture for mobile sensing. *IEEE/ACM Transactions on Networking* 27, 5 (2019), 2043–2055.
[27] Valentin Radu, Nicholas D. Lane, Sourav Bhattacharya, Cecilia Mascolo, Mahesh K. Marina, and Fahim Kawsar. 2016. Towards multimodal deep learning for activity recognition on mobile devices. In *Proceedings of ISWC*. 185–188.
[28] Valentin Radu, Catherine Tong, Sourav Bhattacharya, Nicholas D. Lane, Cecilia Mascolo, Mahesh K. Marina, and Fahim Kawsar. 2018. Multimodal deep learning for activity and context recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 4 (2018), Article 157, 27 pages.
[29] Zhihao Shen, Kang Yang, Wan Du, Xi Zhao, and Jianhua Zou. 2019. DeepAPP: A deep reinforcement learning framework for mobile application usage prediction. In *Proceedings of SenSys*. 153–165.
[30] Hongfei Xue, Wenjun Jiang, Chenglin Miao, Ye Yuan, Fenglong Ma, Xin Ma, Yijiang Wang, et al. 2019. DeepFusion: A deep learning framework for the fusion of heterogeneous sensory data. In *Proceedings of MobiHoc*. 151–160.
[31] Shuochao Yao, Yiran Zhao, Huajie Shao, Aston Zhang, Chao Zhang, Shen Li, and Tarek Abdelzaher. 2018. RDeep Sense: Reliable deep mobile computing models with uncertainty estimations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 4 (2018), Article 173, 26 pages.
[32] Xuan Liu, Xiaoguang Wang, and Stan Matwin. 2018. Improving the interpretability of deep neural networks with knowledge distillation. In *Proceedings of ICDMW*. 905–912.

[33] Liu Ke, Xin He, and Xuan Zhang. 2018. NNest: Early-stage design space exploration tool for neural network inference accelerators. In *Proceedings of ISLPED*. 1–6.
[34] Indranil Palit, Qiuwen Lou, Robert Perricone, Michael Niemier, and X. Sharon Hu. 2019. A uniform modeling methodology for benchmarking DNN accelerators. In *Proceedings of ICCAD*. 1–7.
[35] Vicent Sanz Marco, Ben Taylor, Zheng Wang, and Yehia Elkhatib. 2020. Optimizing deep learning inference on embedded systems through adaptive model selection. *ACM Transactions on Embedded Computing Systems* 19, 1 (2020), 1–28.
[36] Yecheng Xiang and Hyoseung Kim. 2019. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In *Proceedings of RTSS*. 392–405.
[37] A. Gupta, H. P. Gupta, B. Biswas, and T. Dutta. 2020. An early classification approach for multivariate time series of on-vehicle sensors in transportation. *IEEE Transactions on Intelligent Transportation Systems* 21, 12 (2020), 5316–5327.
[38] Ayten Ozge Akmandor, Hongxu Yin, and Niraj K. Jha. 2018. Smart, secure, yet energy-efficient, Internet-of-Things sensors. *IEEE Transactions on Multi-Scale Computing Systems* 4, 4 (2018), 914–930.
[39] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).
[40] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE* 108, 4 (2020), 485–532.
[41] K. Nan, S. Liu, J. Du, and H. Liu. 2019. Deep model compression for mobile platforms: A survey. *Tsinghua Science and Technology* 24, 6 (2019), 677–693.
[42] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems* 13, 3 (2017), 1–18.
[43] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033* (2020).
[44] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of CVPR*. 11264–11272.
[45] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of ICCV*. 1389–1397.
[46] Chongyang Liu and Qinrang Liu. 2018. Improvement of pruning method for convolution neural network compression. In *Proceedings of ICDLT*. 57–60.
[47] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of ICCV*. 2736–2744.
[48] Mengjia Yan, Mengao Zhao, Zining Xu, Qian Zhang, Guoli Wang, and Zhizhong Su. 2019. VarGFaceNet: An efficient variable group convolutional neural network for lightweight face recognition. In *Proceedings of ICCV*. 1–8.
[49] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[50] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. ThiNet: A filter level pruning method for deep neural network compression. In *Proceedings of ICCV*. 5058–5066.
[51] Sourav Bhattacharya and Nicholas D. Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of SenSys*. 176–189.
[52] Yao Zhou, Gary G. Yen, and Zhang Yi. 2021. A knee-guided evolutionary algorithm for compressing deep neural networks. *IEEE Transactions on Cybernetics* 51, 3 (2021), 1626–1638.
[53] Loc N. Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. DeepMon: Mobile GPU-based deep learning framework for continuous vision applications. In *Proceedings of MobiSys*. 82–95.
[54] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of NIPS*. 1269–1277.
[55] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[56] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.
[57] Qing Qin, Jie Ren, Jialong Yu, Hai Wang, Ling Gao, Jie Zheng, Yansong Feng, Jianbin Fang, and Zheng Wang. 2018. To compress, or not to compress: Characterizing deep learning model compression for embedded inference. In *Proceedings of ISPA/IUCC/BDCloud/SocialCom/SustainCom*. 729–736.
[58] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 27–40.

[59] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. 2017. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369* (2017).
[60] Christos Louizos, Karen Ullrich, and Max Welling. 2017. Bayesian compression for deep learning. In *Proceedings of NIPS*. 3288–3298.
[61] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient DNNs. In *Proceedings of NIPS*. 1379–1387.
[62] Xiaoxi He, Zimu Zhou, and Lothar Thiele. 2018. Multi-task zipping via layer-wise neuron sharing. In *Proceedings of NIPS*. 6016–6026.
[63] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of CVPR*. 2820–2828.
[64] Jagmohan Chauhan, Jathushan Rajasegaran, Suranga Seneviratne, Archan Misra, Aruna Seneviratne, and Youngki Lee. 2018. Performance characterization of deep learning models for breathing-based authentication on resource-constrained devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 1–24.
[65] Yiwen Guo, Chao Zhang, Changshui Zhang, and Yurong Chen. 2018. Sparse DNNs with improved adversarial robustness. In *Proceedings of NIPS*. 242–251.
[66] Yong Zhao, Jinyu Li, and Yifan Gong. 2016. Low-rank plus diagonal adaptation for deep neural networks. In *Proceedings of ICASSP*. 5005–5009.
[67] Amir Erfan Eshratifar and Massoud Pedram. 2020. Run-time deep model multiplexing. *arXiv preprint arXiv:2001.05870* (2020).
[68] Jianwei Zhang, Botao Ye, and Xiaohua Luo. 2019. MBFN: A multi-branch face network for facial analysis. In *Proceedings of ACAI*. 542–549.
[69] Huijun Wu, Chen Wang, Jie Yin, Kai Lu, and Liming Zhu. 2018. Sharing deep neural network models with interpretation. In *Proceedings of WWW*. 177–186.
[70] Petko Georgiev, Sourav Bhattacharya, Nicholas D. Lane, and Cecilia Mascolo. 2017. Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), Article 50, 19 pages.
[71] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of CVPR*. 2704–2713.
[72] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.
[73] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yushua Bengio. 2016. Binarized neural networks. In *Proceedings of NIPS*. 4107–4115.
[74] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of FPGA*. 65–74.
[75] Zhenghua Chen, Le Zhang, Zhiguang Cao, and Jing Guo. 2018. Distilling the knowledge from handcrafted features for human activity recognition. *IEEE Transactions on Industrial Informatics* 14, 10 (2018), 4334–4342.
[76] Ji Wang, Weidong Bao, Lichao Sun, Xiaomin Zhu, Bokai Cao, and S. Yu Philip. 2019. Private model compression via knowledge distillation. In *Proceedings of AAAI*. 1190–1197.
[77] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
[78] Sukmin Yun, Jongjin Park, Kimin Lee, and Jinwoo Shin. 2020. Regularizing class-wise predictions via self-knowledge distillation. In *Proceedings of CVPR*. 13876–13885.
[79] Tianhong Li, Jianguo Li, Zhuang Liu, and Changshui Zhang. 2020. Few sample knowledge distillation for efficient network compression. In *Proceedings of CVPR*. 14639–14647.
[80] Xu Cheng, Zhefan Rao, Yilan Chen, and Quanshi Zhang. 2020. Explaining knowledge distillation by quantifying the knowledge. In *Proceedings of CVPR*. 12925–12935.
[81] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. 2019. Improved knowledge distillation via teacher assistant. *arXiv preprint arXiv:1902.03393* (2019).
[82] Chi Nhan Duong, Khoa Luu, Kha Gia Quach, and Ngan Le. 2019. ShrinkTeaNet: Million-scale lightweight face recognition via shrinking teacher-student networks. *arXiv preprint arXiv:1905.10620* (2019).
[83] Jianfei Yang, Han Zou, Shuxin Cao, Zhenghua Chen, and Lihua Xie. 2020. MobileDA: Towards edge domain adaptation. *IEEE Internet of Things Journal* 7, 8 (2020), 6909–6918.

[84] P. Kumari, R. Mishra, and H. P. Gupta. 2021. A knowledge distillation-based transportation system for sensory data sharing using LoRa. *IEEE Sensors Journal* 21, 22 (2021), 25315–25322.
[85] Asit Mishra and Debbie Marr. 2017. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852* (2017).
[86] Guorui Zhou, Ying Fan, Runpeng Cui, Weijie Bian, Xiaoqiang Zhu, and Kun Gai. 2017. Rocket launching: A universal and efficient framework for training well-performing light net. *arXiv preprint arXiv:1708.04106* (2017).
[87] H. Zhao, X. Sun, J. Dong, C. Chen, and Z. Dong. 2022. Highlight every step: Knowledge distillation via collaborative teaching. *IEEE Transactions on Cybernetics* 52, 4 (2022), 2070–2081.
[88] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of MobiSys*. 389–400.
[89] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of MobiCom*. 115–127.
[90] Akhil Mathur, Nicholas D. Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. 2017. DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *Proceedings of MobiSys*. 68–81.
[91] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. DeepCache: Principled cache for mobile deep vision. In *Proceedings of MobiCom*. 129–144.
[92] Alexey Svyatkovskiy, Julian Kates-Harbeck, and William Tang. 2017. Training distributed deep recurrent neural networks with mixed precision on GPU clusters. In *Proceedings of SIGHPC*. 1–8.
[93] Ji Wang, Bokai Cao, Philip Yu, Lichao Sun, Weidong Bao, and Xiaomin Zhu. 2018. Deep learning towards mobile applications. In *Proceedings of ICDCS*. 1385–1393.
[94] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of CVPR*. 6848–6856.
[95] R. Mishra, H. P. Gupta, and T. Dutta. 2021. A road health monitoring system using sensors in optimal deep neural network. *IEEE Sensors Journal* 21, 14 (2021), 15527–15534.
[96] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. 2017. Net-Trim: Convex pruning of deep neural networks with performance guarantee. In *Proceedings of NIPS*. 3177–3186.
[97] Arun Mallya and Svetlana Lazebnik. 2018. PackNet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of CVPR*. 7765–7773.
[98] Shuxin Chen, Lin Lin, Zixun Zhang, and Mitsuo Gen. 2019. Evolutionary NetArchitecture Search for deep neural networks pruning. In *Proceedings of ACAI*. 189–196.
[99] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. 2019. Collaborative channel pruning for deep networks. In *Proceedings of ICML*. 5113–5122.
[100] Pravendra Singh, Vinay Sameer Raja Kadi, Nikhil Verma, and Vinay P. Namboodiri. 2019. Stability based filter pruning for accelerating deep CNNs. In *Proceedings of WACV*. 1166–1174.
[101] Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, et al. 2022. Non-structured DNN weight pruning—Is it beneficial in any platform? *IEEE Transactions on Neural Networks and Learning Systems* 33, 9 (2022), 4930–4944.
[102] Nils T. Siebel, Jonas Botel, and Gerald Sommer. 2009. Efficient neural network pruning during neuro-evolution. In *Proceedings of IJCNN*. 2920–2927.
[103] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Proceedings of NIPS*. 1135–1143.
[104] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. 2020. AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates. In *Proceedings of AAAI*. 4876–4883.
[105] Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of ICLR*. 1–42.
[106] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. 2019. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Proceedings of NIPS*. 1–11.
[107] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. 2020. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of ICML*. 3259–3269.
[108] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. 2020. The early phase of neural network training. In *Proceedings of ICLR*. 1–20.
[109] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G. Baraniuk, Zhangyang Wang, and Yingyan Lin. 2020. Drawing early-bird tickets: Towards more efficient training of deep networks. In *Proceedings of ICLR*. 1–13.
[110] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. 2019. One ticket to win them all: Generalizing lottery ticket initializations across datasets and optimizers. In *Proceedings of NIPS*. 1–11.

[111] Zheng Zhang, Yong Xu, Jian Yang, Xuelong Li, and David Zhang. 2015. A survey of sparse representation: Algorithms and applications. *IEEE Access* 3 (2015), 490–530.
[112] Li Zhang, Wei-Da Zhou, Pei-Chann Chang, Jing Liu, Zhe Yan, Ting Wang, and Fan-Zhang Li. 2011. Kernel sparse representation-based classifier. *IEEE Transactions on Signal Processing* 60, 4 (2011), 1684–1695.
[113] Deepak Kadetotad, Sairam Arunachalam, Chaitali Chakrabarti, and Jae-Sun Seo. 2016. Efficient memory compression in deep neural networks using coarse-grain sparsification for speech applications. In *Proceedings of ICCAD*. 1–8.
[114] Jacob Søgaard Larsen and Line Clemmensen. 2020. Weight sharing and deep learning for spectral data. In *Proceedings of ICASSP*. 4227–4231.
[115] Shubham Jain, Swagath Venkataramani, Vijayalakshmi Srinivasan, Jungwook Choi, Pierce Chuang, and Leland Chang. 2018. Compensated-DNN: Energy efficient low-precision deep neural networks by compensating quantization errors. In *Proceedings of DAC*. 1–6.
[116] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).
[117] Zhiqiang Tang, Xi Peng, Kang Li, and Dimitris N. Metaxas. 2020. Towards efficient U-Nets: A coupled and quantized approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 8 (2020), 2038–2050.
[118] Mary Phuong and Christoph Lampert. 2019. Towards understanding knowledge distillation. In *Proceedings of ICML*. 5142–5151.
[119] Yuang Liu, Wei Zhang, and Jun Wang. 2022. Multi-knowledge aggregation and transfer for semantic segmentation. In *Proceedings of AAAI*. 1–9.
[120] Fahad Sarfraz, Elahe Arani, and Bahram Zonooz. 2021. Knowledge distillation beyond model compression. In *Proceedings of ICPR*. 6136–6143.
[121] Fei Yuan, Linjun Shou, Jian Pei, Wutao Lin, Ming Gong, Yan Fu, and Daxin Jiang. 2021. Reinforced multi-teacher selection for knowledge distillation. In *Proceedings of AAAI*, Vol. 35. 14284–14291.
[122] Sajjad Abbasi, Mohsen Hajabdollahi, Nader Karimi, and Shadrokh Samavi. 2020. Modeling teacher-student techniques in deep neural networks for knowledge distillation. In *Proceedings of MVIP*. 1–6.
[123] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Proceedings of NIPS*. 137–144.
[124] SHL Challenge. 2022. Home Page. Retrieved November 11, 2022 from http://www.shl-dataset.org/activity-recognition-challenge/.
[125] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of CVPR*. 4510–4520.
[126] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *Proceedings of ECCV*. 116–131.
[127] Saideep Tiku, Prathmesh Kale, and Sudeep Pasricha. 2021. QuickLoc: Adaptive deep-learning for fast indoor localization with mobile devices. *ACM Transactions Cyber-Physical Systems* 5, 4 (2021), Article 44, 30 pages.
[128] Liping Wang, Saideep Tiku, and Sudeep Pasricha. 2022. CHISEL: Compression-aware high-accuracy embedded indoor localization with deep learning. *IEEE Embedded Systems Letters* 14, 1 (2022), 23–26.
[129] Thomas Altstidl, Sebastian Kram, Oskar Herrmann, Maximilian Stahlke, Tobias Feigl, and Christopher Mutschler. 2021. Accuracy-aware compression of channel impulse responses using deep learning. In *Proceedings of IPIN*. 1–8.
[130] Guanzhong Tian, Jun Chen, Xianfang Zeng, and Yong Liu. 2021. Pruning by training: A novel deep neural network compression framework for image processing. *IEEE Signal Processing Letters* 28 (2021), 344–348.
[131] Reagan L. Galvez, Argel A. Bandala, Elmer P. Dadios, Ryan Rhay P. Vicerra, and Jose Martin Z. Maningo. 2018. Object detection using convolutional neural networks. In *Proceedings of TENCON*. 2023–2027.
[132] Andrey V. Savchenko. 2021. Facial expression and attributes recognition based on multi-task learning of lightweight neural networks. In *Proceedings of SISY*. 119–124.
[133] Yu Zhao, Yue Yin, and Guan Gui. 2020. Lightweight deep learning based intelligent edge surveillance techniques. *IEEE Transactions on Cognitive Communications and Networking* 6, 4 (2020), 1146–1154.
[134] Hang-Rui Hu, Yan Song, Ying Liu, Li-Rong Dai, Ian McLoughlin, and Lin Liu. 2022. Domain robust deep embedding learning for speaker recognition. In *Proceedings of ICASSP*. 7182–7186.
[135] Mario Almeida, Stefanos Laskaridis, Abhinav Mehrotra, Lukasz Dudziak, Ilias Leontiadis, and Nicholas D. Lane. 2021. Smart at what cost? Characterising mobile deep neural networks in the wild. In *Proceedings of IMC*. 658–672.
[136] Shayan Hassantabar, Joe Zhang, Hongxu Yin, and Niraj K. Jha. 2022. MHDeep: Mental health disorder detection system based on wearable sensors and artificial neural networks. *ACM Transactions on Embedded Computing Systems*. Accepted March 2022.
[137] Jinyang He, Jiqing Chen, Jun Liu, and Hengyu Li. 2019. A lightweight architecture for driver status monitoring via convolutional neural networks. In *Proceedings of ROBIO*. 388–394.

[138] Huy Phan, Yi Xie, Jian Liu, Yingying Chen, and Bo Yuan. 2022. Invisible and efficient backdoor attacks for compressed deep neural networks. In *Proceedings of ICASSP*. 96–100.
[139] Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. 2019. DeepAttest: An end-to-end attestation framework for deep neural networks. In *Proceedings of ISCA*. 487–498.
[140] Swati Chopade, Hari Prabhat Gupta, Rahul Mishra, Aman Oswal, Preti Kumari, and Tanima Dutta. 2022. A sensors based river water quality assessment system using deep neural network. *IEEE Internet of Things Journal* 9, 16 (2022), 14375–14384.
[141] Adrián Alcolea and Javier Resano. 2022. Bayesian neural networks to analyze hyperspectral datasets using uncertainty metrics. *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022), 1–10.
[142] Jalel Ktari, Tarek Frikha, Monia Hamdi, Hela Elmannai, and Habib Hmam. 2022. Lightweight AI framework for Industry 4.0 case study: Water meter recognition. *Big Data and Cognitive Computing* 6, 3 (2022), 72.
[143] Chitranjan Singh, Rahul Mishra, Hari Prabhat Gupta, and Preti Kumari. 2022. The Internet of Drones in precision agriculture: Challenges, solutions, and research opportunities. *IEEE Internet of Things Magazine* 5, 1 (2022), 180–184.
[144] Christos Kyrkou, George Plastiras, Theocharis Theocharides, Stylianos I. Venieris, and Christos-Savvas Bouganis. 2018. DroNet: Efficient convolutional neural network detector for real-time UAV applications. In *Proceedings of DATE*. 967–972.
[145] Xin Yang, Jingyu Chen, Yuanjie Dang, Hongcheng Luo, Yuesheng Tang, Chunyuan Liao, Peng Chen, and Kwang-Ting Cheng. 2021. Fast depth prediction and obstacle avoidance on a monocular drone using probabilistic convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems* 22, 1 (2021), 156–167.
[146] Maria Tzelepi and Anastasios Tefas. 2021. Graph embedded convolutional neural networks in human crowd detection for drone flight safety. *IEEE Transactions on Emerging Topics in Computational Intelligence* 5, 2 (2021), 191–204.

Received 10 January 2022; revised 13 October 2022; accepted 25 October 2022