Deep Learning on Mobile and Embedded Devices: State-of-the-art, Challenges, and Future Directions

YANJIAO CHEN, BAOLIN ZHENG, ZIHAN ZHANG, and QIAN WANG,
Wuhan University, China
CHAO SHEN, Xi'an Jiaotong University, China
QIAN ZHANG, Hong Kong University of Science and Technology, China

Recent years have witnessed an exponential increase in the use of mobile and embedded devices. With the great success of deep learning in many fields, there is an emerging trend to deploy deep learning on mobile and embedded devices to better meet the requirement of real-time applications and user privacy protection. However, the limited resources of mobile and embedded devices make it challenging to fulfill the intensive computation and storage demand of deep learning models. In this survey, we conduct a comprehensive review on the related issues for deep learning on mobile and embedded devices. We start with a brief introduction of deep learning and discuss major challenges of implementing deep learning models on mobile and embedded devices. We then conduct an in-depth survey on important compression and acceleration techniques that help adapt deep learning models to mobile and embedded devices, which we specifically classify as pruning, quantization, model distillation, network design strategies, and low-rank factorization. We elaborate on the hardware-based solutions, including mobile GPU, FPGA, and ASIC, and describe software frameworks for mobile deep learning models, especially the development of frameworks based on OpenCL and RenderScript. After that, we present the application of mobile deep learning in a variety of areas, such as navigation, health, speech recognition, and information security. Finally, we discuss some future directions for deep learning on mobile and embedded devices to inspire further research in this area.

CCS Concepts: • **Networks** → **Mobile networks**; • **Computing methodologies** → **Machine learning**; • **Software and its engineering** → **Designing software**; • **Hardware** → *Wireless devices*; *Reconfigurable logic and FPGAs*;

Additional Key Words and Phrases: Deep learning, mobile devices, network compression and acceleration, hardware solutions, software frameworks

Yanjiao's research is supported by the National Natural Science Foundation of China under Grants 61972296 and 61702380, and Wuhan Advanced Application Project under Grant 2019010701011419. Qian's research is supported by the National Natural Science Foundation of China under Grants 61822207 and U1636219, the Equipment Pre-research Joint Fund of Ministry of Education of China (Youth Talent) under Grant 6141A02033327, and the Outstanding Youth Foundation of Hubei Province under Grant 2017CFA047.

Authors' addresses: Y. Chen, B. Zheng, Z. Zhang, and Q. Wang (corresponding author), Wuhan University, Wuhan, 40072, China; emails: {chenyanjiao, baolinzheng, zhangzihan, qianwang}@whu.edu.cn; C. Shen, Xi'an Jiaotong University, Xi'an, 710049, China; email: chaoshen@xjtu.edu.cn; Q. Zhang, Hong Kong University of Science and Technology, Hong Kong, China; email: qianzh@cse.ust.hk.

0360-0300/2020/08-ART84 $15.00
https://doi.org/10.1145/3398209

ACM Reference format:
Yanjiao Chen, Baolin Zheng, Zihan Zhang, Qian Wang, Chao Shen, and Qian Zhang. 2020. Deep Learning on Mobile and Embedded Devices: State-of-the-art, Challenges, and Future Directions. *ACM Comput. Surv.* 53, 4, Article 84 (August 2020), 37 pages.
https://doi.org/10.1145/3398209

1 INTRODUCTION

Recent years have witnessed the rapid development of deep learning with successful applications in a wide range of areas, including image classification [91, 169], speech recognition [33], object detection [149], semantic segmentation [113], and natural language processing (NLP) [125]. Meanwhile, mobile and embedded devices are becoming more and more popular, including smartphones, smart glasses, and smartwatches. These smart and embedded devices are usually equipped with a variety of artificial intelligence applications, such as Apple Siri, to help human-computer interactions. As shown in Figure 1, there are mainly two modes of mobile deep learning, i.e., online mode and offline mode. Earlier efforts on leveraging deep learning models for mobile applications mainly focused on the online mode, i.e., the cloud performs the training and inference tasks while mobile devices only send and receive data from the cloud due to their limited battery, computation power, and storage. However, this online mode depends on Internet connectivity, which may induce a long delay. Furthermore, outsourcing deep learning tasks to a remote server may bring privacy concerns about users' sensitive data. In comparison, in the offline mode, the training task is still performed by the cloud, but the trained model is sent to the mobile device to conduct inference locally (edge-side inference) to preserve user privacy. However, the trained deep models may have a large number of parameters and require complicated computations, which pose great challenges for the limited resources on mobile devices. Thanks to the advent of powerful mobile and embedded devices with dedicated hardware accelerators, multi-core processors and gigabytes RAM, there is an emerging trend to leverage the offline mode to directly deploy deep learning models on mobile and embedded devices for inference. Compared with the online mode, the offline mode can better protect user privacy and greatly reduce response time, communication costs, and cloud burdens. Due to its availability, reliability, security, and low latency, the offline mode is the preferred mode for various applications, especially for real-time applications.

Unfortunately, the computation power, memory, memory bandwidth, and battery of mobile and embedded devices are too limited to support modern deep learning models that are both computing-demanding and memory-demanding, as shown in Table 1. For instance, the ResNet-50 model has 25.6M parameters, taking up more than 87 MB storage space, and needs 3.8B floating-point operations to classify a single image. As the model accuracy improves with the scale of the neural network, the amount of computation and storage involved in deep learning models may be unbearable for mobile and embedded devices even during the inference phase. Besides, the excessive energy consumption from performing deep learning models on battery-powered mobile devices is also a severe problem to be solved. Moreover, diverse computing environments of mobile and embedded devices present additional challenges for mobile deep learning.

In this article, we present a comprehensive survey on state-of-the-art research of deep learning on mobile and embedded devices. We are especially interested in solutions that enable efficient and effective deployment of deep learning models on various mobile and embedded devices. Specifically, we present existing solutions from three aspects: compression & acceleration techniques, hardwares, and software frameworks.

The vast number of parameters of deep learning models are quite redundant, making it possible to compress and accelerate deep learning models without significantly degrading the accuracy,

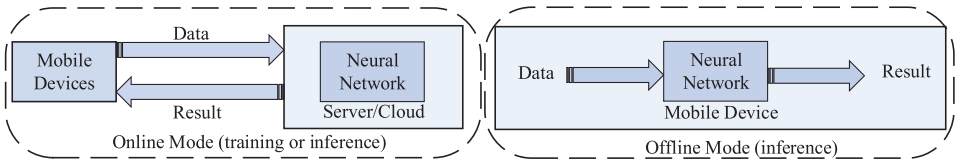


Fig. 1. Two modes of mobile deep learning.

Table 1. The Comparison of Typical Deep Learning Models

Model	Layer	Top-1/Top-5 Acc(%) on ImageNet	Parameters			Computation		
			Size(M/MB)	Conv(%)	Fc(%)	FLOPs(G)	Conv(%)	Fc(%)
AlexNet [91]	8	57.0/80.3	61/238	3.8	96.2	0.72	91.9	8.1
VGG-16 [160]	16	70.5/90.0	138/528	10.6	89.4	15.5	99.2	0.8
GoogLeNet [169]	22	72.5/90.8	6.9/90	85.1	14.9	1.6	99.9	0.1
ResNet-18 [64]	18	70.7/89.9	11.2/44	96.4	4.6	1.8	99.9	0.1
ResNet-50 [64]	50	75.8/92.9	25.6/87	92.0	8.0	3.8	99.9	0.1
MobileNet [71]	28	70.4/89.5	4.2/16	97.6	2.4	0.57	99.8	0.2

as shown in Table 1. There have been extensive studies in network compression and acceleration techniques to achieve this goal, including pruning, quantization, model distillation, network design strategies, and low-rank factorization. Besides, designing hardware accelerators is desirable to speed up neural networks, since deep learning models pose high pressure on mobile hardware. Apart from traditional hardware such as CPU and GPU, semi-customized and customized chips such as Field-Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) can enable an accelerator to achieve a higher speed and lower energy consumption, which inspire a lot of FPGA/ASIC-based accelerators for mobile and embedded devices. Recent works tend to exploit parallelism and improve energy efficiency on the hardware fully. Furthermore, deep learning software frameworks are popular among developers and researchers, because off-the-shelf infrastructures and building blocks are provided to avoid building models from scratch. But there is a need to redevelop the software framework for mobile deep learning, as many desktop frameworks are not available for a variety of mobile and embedded devices.

Though still in its infancy, mobile deep learning has been applied to many areas. For example, navigation applications equipped with deep learning models can give real-time instructions on directions, obstacles, and GPS locations [6, 87, 208]. Deep learning–enabled image processing can help recognize food and pills for health management [124, 173, 196]. Audio sensing–based on deep learning models can enable various mobile applications, e.g., speech recognition, user identification, and emotion detection [44, 120, 145]. With the advances in the solutions for mobile deep learning, we expect there will be booming opportunities to develop deep learning–based mobile applications.

This article provides a more comprehensive and in-depth survey on mobile deep learning compared to existing surveys on similar topics, including most recent research advances. For example, References [3, 55, 92] only surveyed deep learning on FPGAs. Reference [25] briefly introduced early works on compression and acceleration methods for deep neural network models. References [34, 63, 144] concerned about deep learning in general but not focused on deep learning on mobile and embedded devices, which are significantly different from our work. References [24, 138] described mobile deep learning but with a much less breadth and depth compared to our work. Reference [168] mainly focused on hardware solutions, such as hardware platforms, circuits

design, and the use of new memory technologies. Reference [198] discussed the recent application of deep learning to mobile and wireless networking, with only a brief description of how to optimize deep learning models for mobile devices. Reference [127] surveyed the usage of deep learning in the IoT domain, but the discussion on adapting deep learning models to suit mobile devices is brief. To the best of our knowledge, we are the first to present a comprehensive survey on state-of-the-art research of mobile deep learning, introducing the most-recent research progress from the aspects of compression & acceleration techniques, hardwares, software frameworks, and applications.

The article is organized as follows: In Section 2, we give a brief introduction of deep learning and discuss the challenges of deploying deep learning models on mobile and embedded devices. Section 3 presents a comprehensive review of the compression and acceleration techniques for mobile deep learning models. An overview of hardware solutions is outlined in Section 4. In Section 5, software frameworks for mobile deep learning models are described. We introduce applications of deep learning on mobile and embedded devices in Section 6. Privacy issues and potential future directions are discussed in Section 7 and Section 8, respectively. We conclude the article in Section 9.

This article surveys the literature over the period 2000–2018 on deep learning on mobile and embedded devices.

2 BACKGROUND

In this section, we give a brief introduction of deep learning and present major challenges of deploying deep learning on mobile devices.

2.1 Deep Learning

Deep learning refers to large-scale machine learning algorithms inspired by connectionism and biological nervous systems. Thanks to the availability of powerful GPUs at affordable prices, in recent years, deep learning has made unprecedented progress in a wide range of tasks, e.g., image classification, speech recognition, and natural language processing. Compared with traditional machine learning algorithms that rely largely on extracted features defined by human developers, deep learning can automatically extract features from the raw data. In this way, deep learning can outperform traditional machine learning algorithms by a great margin, making a great success in the field of artificial intelligence and attracting significant attention from both industry and academia. There are two phases in deep learning: the training phase and the inference phase. The training data are used to train learning models in the training phase, and the trained model is used to predict results with the input data in the inference phase.

There is tremendous progress in deep learning architectures, the most popular of which are Deep Neural Network (DNN), Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN). Deep Neural Network (DNN) usually consists of the input layer, multiple nonlinear hidden layers, and the output layer, as shown in Figure 2. Activation functions are applied to the calculated value to add nonlinearity to the network. During each epoch in the training phase, weight parameters of each layer will be updated by back-propagation algorithm [152]. According to the processing method, the DNNs can be divided into feedforward neural networks, feedback neural networks, and self-organizing neural networks.

CNN is a class of DNN, inspired by the behavior of visual cortexes, which is popular in the field of image and video processing. CNN features share weights and translation invariance. As shown in Figure 3, CNN consists of convolutional layers, pooling layers, and fully connected layers. Convolutional layers contain weights of a set of filters (or kernels) to be shared with the previous layer. Typical models of CNNs include LeNet, AlexNet, VGG, GoogleNet, and ResNet.

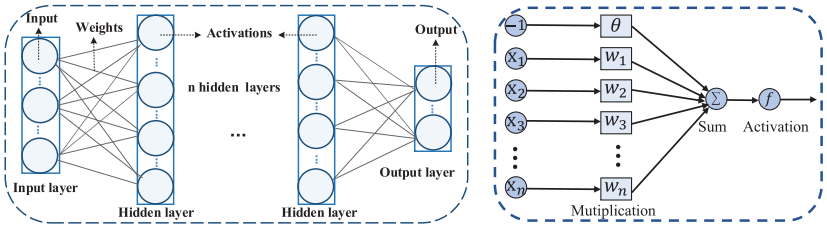
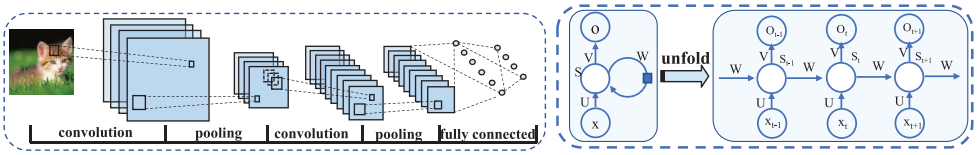


Fig. 2. A typical example of Deep Neural Network (left) and the detail of one neuron unit in Deep Neural Network (right).



Convolutional Neural Network Recurrent Neural Network

Fig. 3. A typical example of Convolutional Neural Network (left) and Recurrent Neural Network (right).

Recurrent Neural Networks (RNNs) show great promise in many NLP and audio processing tasks, using sequential information based on the directed graph built upon the connections between nodes, as shown in Figure 3. Long Short-Term Memory (LSTM) models, consisting of forget gates, input gates, and output gates, are useful and commonly used RNNs.

2.2 Challenges

Recent years have witnessed rapid development and huge success of deep learning, which have led to breakthroughs in many fields of artificial intelligence. The accuracy of DNN models consistently improves with the increasing scales. At the same time, with the deep penetration of mobile devices, there is a strong driving force to deploy DNNs in mobile applications.

There are three major challenges in deploying deep learning in mobile and embedded devices.

- *Diverse computing environment.* The hardware and software environment of mobile and embedded devices is very different from that of computers. Therefore, the current deep learning models developed for computers cannot be directly applied to mobile and embedded devices. For example, many CUDA-based libraries for GPU acceleration are not available in mobile GPUs that have less space and power than traditional desktop GPUs.
- *Limited resources.* The dense computation inherent in DNNs will consume a substantial quantity of resources, including battery power, memory, and computation units, which are limited on mobile devices. Currently, as shown in Table 1, a standard DNN contains dozens of convolution layers, which demands a considerable storage space and computation power that may not be affordable for mobile and embedded devices during either the training phase or the inference phase. Moreover, the prohibitive energy consumption of running deep learning algorithms also poses a significant challenge for battery-powered mobile devices.
- *Long delay.* Due to its complicated structure and the limited computation power of mobile devices, the delay of the inference phase of deep learning models may be long, which may not be able to meet the requirement of real-time applications, like autonomous vehicles.

Table 2. A Brief Summary of Network Compression and Acceleration Techniques

Technique	Description	Key problem	Characteristics
Pruning	Removing low-saliency parameters	Evaluation metrics	Widely used, good performance
Quantization	Reducing the number of bits for parameters representation	Accuracy reduction	Widely used, hardware-friendly
Model distillation	Training a distilled student model that mimics a larger teacher network	Definition of transferred knowledge	Train from scratch, popular in NLP
Network design strategies	Designing low-cost and efficient architectures	Proper strategies	Train from scratch, almost exclusive for CNN
Low-rank factorization	Using approximate low-rank tensors	Decomposition method	Not popular in small filters

Additionally, the input data of some applications may be huge, e.g., video processing, which requires a large amount of memory and takes a long time to process.

To address these difficulties, many great ideas and works have been proposed over the past few years, including network compression and acceleration techniques, as well as hardwares and software frameworks, which will be introduced in detail in following sections.

3 NETWORK COMPRESSION AND ACCELERATION TECHNIQUE

Due to the high computation cost of deep learning models and the resource constraint of mobile and embedded devices, it is hard to directly deploy deep learning models on mobile and embedded devices. Thanks to the fact that some parameters of deep learning models may be redundant, we can compress and accelerate the models without significantly degrading the performance.

The network compression and acceleration techniques have attracted a lot of attention from both industry and academia. We mainly classify these methods into seven categories: pruning, quantization, model distillation, network design strategies, low-rank factorization, and other techniques as well as hybrid techniques. The pruning technique eliminates redundant parameters by evaluating their contribution to model performance. Different metrics, which remain a significant challenge for pruning techniques, as shown in Table 2, have been proposed to assess the importance of parameters. Quantization techniques use a fewer number of bits to represent parameters (e.g., binary/ternary), which helps reduce memory overhead and computation time. The model distillation technique is based on student-teacher models, which trains a smaller distilled student model based on transferred knowledge from a more extensive teacher network. Network design strategies explore efficient strategies for designing special blocks to obtain low-cost architecture. The low-rank factorization technique uses tensor decomposition to compress and accelerate networks. Moreover, these techniques are orthogonal to others and can be integrated to achieve notable performance. We briefly summarize these network compression and acceleration techniques in Table 2.

3.1 Pruning

Pruning is a very popular model compression and acceleration technique that compresses the network by pruning unimportant and inefficient parameters. A typical pruning algorithm can be divided into three stages: evaluating the importance of parameters, pruning unimportant parameters, and fine-tuning to recover accuracy, as shown in Figure 4.

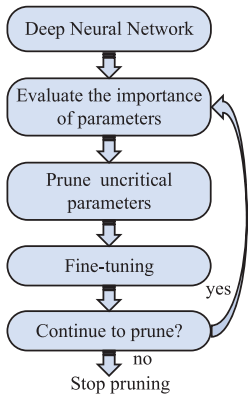


Fig. 4. An illustration of typical pruning methods. Three stages in each iteration: evaluating the importance of parameters, pruning unimportant parameters, and fine-tuning to recover accuracy.

LeCun et al. [97] first demonstrated that some unsalient weights could be eliminated from pre-trained networks without affecting accuracy. However, such a non-structured pruning technique may lead to an irregular structure, which cannot be accelerated directly, and the non-structured random connectivity may give rise to cache and memory access issues. To address these limitations, structured pruning is proposed to obtain regular network connections. Therefore, we classify the pruning technique into two categories: non-structured pruning and structured pruning. As shown in Table 3, we compare different existing pruning techniques in terms of the technique, dataset, model, compression rate, acceleration rate, and the accuracy drop.

3.1.1 Non-structured Pruning. Han et al. [62] first introduced a layer-wise magnitude-based pruning technique that iteratively eliminated the weights that are below the threshold and re-trained the model to recover accuracy. Guo et al. [56] proposed dynamic network surgery that added a splicing operation to recover the critical but pruned connection after pruning. Zhu et al. [213] demonstrated that small-dense networks could not be compared with large-sparse counterparts and proposed a gradual pruning technique that zeros the weights with the smallest magnitude. Li et al. [101] demonstrated that tuning the layer-specific thresholds is difficult and proposed an automatic layer-wise magnitude-based pruning that transforms the threshold tuning problem into a constrained optimization problem. Moreover, Srinivas et al. [164] considered that similar neurons could be eliminated due to redundancy.

The pruning technique is effective not only in the spatial domain but also in other domains. Liu et al. [108] combined Winograd's minimal filtering algorithm with network pruning and proposed two modifications to Winograd-based CNNs to attain sparsity without lowering the accuracy, where the ReLU operation and the pruning operation were moved into the Winograd domain. Similarly, Liu et al. [111] proposed frequency-domain dynamic pruning schemes that pruned spatial correlations after 2D DCT transformation.

Yang et al. [190] argued that the metrics of model size, computation, and energy consumption in network compression and acceleration techniques were not necessarily consistent due to data movement. Previous works have focused more on model size and computation but not energy consumption. Hence, Yang et al. presented energy-aware pruning (EAP) that used the framework proposed in Eyeriss [23] to estimate the energy consumption of Multiply-and-Accumulate (MAC) operations and data movements on actual hardware. Additionally, Yang et al. [189] proposed an end-to-end network energy-constrained framework that leveraged input masking to increase input sparsity and knowledge distillation to help optimization.

Table 3. A Comparison of Different Pruning Techniques

	Technique	Dataset	Model	Compression	Acceleration	Top-5 acc drop
Non-Structured pruning	Magnitude-based pruning [62]	ImageNet	AlexNet	9×	-	−0.45%
	Gradual pruning [213]	ImageNet	InceptionV3	4.0×	-	1.1%
	Dynamic network surgery [56]	ImageNet	AlexNet	17.7×	-	−0.13%
	OLMP [101]	Caltech-256	AlexNet-Caltech	82×	-	0.4%
	Removing similar neurons [164]	ImageNet	AlexNet	2.9×	-	2.24%
	Winograd domain pruning [108]	ImageNet	ResNet-18 variation	2.8×	10.8×	0.07%
	Frequency domain pruning [111]	ImageNet	AlexNet	22.6×	-	−0.08%
	Energy aware pruning [190]	ImageNet	AlexNet	11.1×	3.7×*	0.8%
	Energy-constrained pruning [189]	ImageNet	AlexNet	3.2×	3.8×*	0.5%
	RPP [53]	SST-2	BERT	2 ×	-	1.9%
Structured pruning	IKSS [7]	CIFAR-10	CIFAR-10 Network	3.3×	-	1%
	SSL [184]	ImageNet	AlexNet	-	5.1×	2.03%
	BN scaling factor [109]	ImageNet	VGG-A	6.6×	1.43×	0.03%
	APoZ [72]	ImageNet	VGG-16	2.59×	-	−1.35%
	reconstruction-based pruning [67]	ImageNet	VGG-16, ResNet-50	-	4.4×, 2.83×	2.28%, 2.07%
	AutoPruner [115]	ImageNet	VGG-16, ResNet-50	-	3.79 ×, 2.92×	1.95%, 1.76%
	DCP [214]	ImageNet	ResNet-50	2.06×	2.25×	0.61%
	Absolute weight sum [102]	ImageNet	VGG-16	-	2×	0.8%
	TP [130]	ImageNet	VGG-16	-	2.68×	3.38%
	Gradient-based pruning [54]	CIFAR-10	NIN	-	1.8×	3%
	ThiNet [114]	ImageNet	VGG-16, ResNet-50	16.63×, 2.06×	3.3×, 2.26×	0.52%, 1.12%
	D-Pruner [112]	CIFAR-10	NIN	2.77×	2.05×	0.56%
	GDP [106]	ImageNet	VGG-16, ResNet-50	-	2.42×, 1.96×	0.65%, 1.59%
	SFP [65]	ImageNet	ResNet-101	-	1.73×	−0.20%
	Auto-balanced pruning [37]	CIFAR-10	VGG-16, ResNet-56	-	5.37×, 2.56×	0.38%, 0.1%
	IncReg [178]	ImageNet	VGG-16, ResNet-50	-	4.0×, 2.0×	0.8%, 0.1%
	Prune decision [209]	CIFAR-10	VGG-19, ResNet-56	-	6.5×, 1.91×	0.36%, 0.11%

*indicates the rate of energy saving rather than acceleration.

To avoid the undesirable effects of adding regularization penalty items directly to the loss function, Reweighted Proximal Pruning (RPP) [53] utilized proximal algorithm to separate sparsity pattern search from the gradient-based update. Reweighted L_1 minimization was integrated with proximal algorithm to prune large-scale language representation models, such as BERT [36].

3.1.2 Structured Pruning. Unstructured pruning usually leads to irregular network structure that may not fit well in parallel computation and demand extra representation efforts. Structured sparsity is better for computation resource savings on mobile and embedded devices. The granularity of structured sparsity achieved in different structured pruning methods is different. According to the granularity of structured sparsity, we classify structured pruning into two groups: vector-level & kernel-level pruning, and channel-level & filter-level pruning [118].

Vector-level & Kernel-level Pruning. Vector-level & kernel-level pruning techniques prune the vectors in the convolutional kernels and the convolutional kernels in a structured way. Anwar et al. [7] first introduced the kernel-level pruning and intra-kernel strided pruning that prunes the kernel in a fixed stride. Mao et al. [118] explored structured pruning at various granularities and

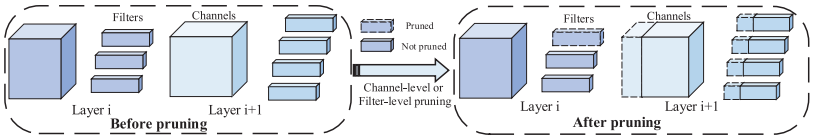


Fig. 5. An illustration of channel-level and filter-level pruning.

demonstrated that structured pruning outperformed non-structured pruning in terms of network acceleration and compression due to the advantage of regular sparsity and index saving. Structured Sparsity Learning (SSL) [184] regularized the structures of DNNs in terms of kernels, filters, filter shapes, and layer depth. Unlike previous works, SSL did not rely on the pre-trained networks and directly trained a compact network by group Lasso regularization from scratch.

Channel-level & Filter-level Pruning. The channel-level pruning technique prunes the channels in the feature maps, and the filter-level pruning technique prunes an entire convolutional filter. Channel-level & filter-level pruning can significantly reduce the computation cost and memory overhead. In addition, the off-the-shelf libraries are available for the architecture of channel-level & filter-level pruning due to its intact structure. As shown in Figure 5, a channel in the feature map is pruned. Thus, the corresponding filter in the previous layer can be removed, as well as the corresponding channel of filters in the next layer. Hence, channel-level pruning is consistent with filter-level pruning in terms of pruning granularities.

The core of channel-level & filter-level pruning technique is to identify the importance of the channels and filters, respectively. Based on the assumption that the amount of information contributed by each channel can be evaluated by the channel activation output's variance, variances-based criteria [143] pruned channels according to their contribution variance. Network slimming [109] imposed $L1$ regularization on the scaling factor in batch normalization (BN) layers to train compact networks from scratch, driving its value towards zero to identify insignificant channels (or neurons). Hu et al. [72] leveraged Average Percentage of Zeros (APoZ) of channels (or neurons) on a large dataset to estimate the saliency of each channels (or neurons). Besides, He et al. [67] utilized the LASSO regression to select important channels of input feature maps and least square to reconstruct the output.

It is critical to evaluate the importance of channels so a new neural network can be used. Luo et al. [115] proposed an efficient channel selection layer AutoPruner, where channel pruning and fine-tuning were jointly exploited to identify unimportant channels. Discrimination-aware channel pruning (DCP) [214] assumed that an informative channel has discriminative power. Thus discrimination-aware losses are added to improve the intermediate layers' discriminative power. The impact of channel selection on performance should also be considered. Radu et al. [147] found that pruning several channels may elongate instead of shrink the inference time. The optimal amount of channels depends on the job dispatches on the kernels of libraries.

There is also a line of work on filter-level pruning in recent years. Similar to non-structured pruning, magnitude-based pruning technique [102] calculated absolute weight sum of each filter to remove unimportant filters. Molchanov et al. [130] utilized Taylor expansion to calculate the approximated change of the cost function when removing each filter. A gradient-based technique [54] aimed to identify the influence of specific filters on the given classes based on the assumption that different filters represent different features. ThiNet [114] pruned filters using the next layer's statistics information instead of the current layer's. By minimizing reconstruction errors of the feature map, the channels are selected greedily. Similar to channel-level pruning, D-Pruner [112] expanded convolutional layers as masking blocks to evaluate the impact of each filter on the accuracy.

Pruning the filters in a fixed way may cause severe performance loss. A Global & Dynamic Pruning (GDP) scheme [106] was proposed that globally pruned unimportant filters based on a global discriminative function. Furthermore, filters and the global mask are updated dynamically and iteratively to enhance the performance. Similar to dynamic network surgery [56], Soft Filter Pruning (SFP) [65] was proposed to solve the problem of model capacity reduction and irretrievable network damage. The SFP dynamically prunes the filters in a soft manner to recover critical filters and can train a model from scratch.

Previous regularization-based pruning techniques usually have a large and constant regularization factor to drive weights towards zero, which may give rise to two problems. First, it is improper to treat all weights in different groups as equally important. Second, the expressiveness of CNNs may be degraded. To address the first problem, Ding et al. [37] proposed an auto-balanced filter pruning (AFP) pipeline to punish unsalient filters more and roll back the irreversibly damaged network to the previous state. Furthermore, Wang et al. [178] introduced incremental regularization, where different regularization factors were incrementally assigned to various groups. Zhong et al. [209] argued that the pruning order of the layers was important and utilized a Long Short-Term Memory (LSTM) to guide the pruning order.

Liu et al. [110] observed that structured pruning (i.e., inheriting weights from pre-trained model and fine-tuning the pruned model) may be worse than training pruned networks from scratch, which demonstrated that pruning techniques could be regarded as performing network architecture search. Therefore, apart from evaluating the importance of weights, it is also essential to search for the optimal network architecture for the pruned model.

3.2 Quantization

Network quantization is a mature compression and acceleration technique that has a long history. We can mainly classify quantization techniques into two groups: linear quantization and non-linear quantization. The key idea of linear quantization is to reduce the number of bits that represent each activation or weight, while the key idea of non-linear quantization is to divide weights into a few groups, and each group shares a single weight. The primary concern of quantization techniques is to maintain accuracy as much as possible. As shown in Table 4, we briefly summarize different existing quantization methods.

3.2.1 Linear Quantization & Fixed-point Quantization. Using 32-bit floating-point numbers to represent weights or activations will consume a large number of resources. Hence, linear quantization, as refer to fixed-point quantization, utilizes the low-bit fixed-point number representation to approximate each weight and activation. The extreme case is binary weight networks that represent weight by only 1 bit, i.e., +1 or −1. Ternary weight network is also proposed to achieve a good performance, where a weight is represented by three values, i.e., +1, 0, or −1. Ternary weight networks can also be viewed as non-linear techniques if the ternary values are trainable. Linear quantization is a hardware-friendly and mature technique that is commonly adopted in hardware accelerator design. We will introduce techniques of linear quantization in terms of weight quantization and activation quantization.

Linear Quantization of Weights. Suyog et al. [57] argued that weights of deep networks could be represented by 16-bit fixed-point numbers without significantly reducing classification accuracy. Courbariaux et al. [30] analyzed the impact of three weight representation formats on the final error, such as floating point, fixed point, and dynamic fixed point. The results showed that dynamic fixed point was superior to the other two formats in the training of DNNs. Furthermore, Ristretto [58] was proposed to condense the AlexNet to 8-bit. Mathew et al. [119] presented a quick sparsity and quantization technique using an 8-bit fixed point by modifying the Caffe CNN

Table 4. A Comparison of Different Quantization Methods

	Technique	Weight	Activation	Dataset	Model	Top-5 acc drop
Weight quantization	Fixed point [57]	16-bit	16-bit	CIFAR-10	4-layer CNN	0.8%
	Dynamic fixed point [30]	10-bit	10-bit	CIFAR-10	Maxout	3.14%
	Ristretto [58]	8-bit	8-bit	ImageNet	AlexNet	0.6%
	EBP [26]	Binary	32-bit	MNIST	MNN	0.46%
	BinaryConnect [31]	Binary	32-bit	ImageNet	AlexNet	19.2%
	BWN [148]	Binary	32-bit	ImageNet	AlexNet, ResNet-18	0.8%, 6.2%
	BWNH [73]	Binary	32-bit	ImageNet	AlexNet	−0.7%
	TWN [100]	Ternary	32-bit	ImageNet	AlexNet, ResNet-18	3.5%, 3%
	TTQ [212]	Ternary	32-bit	ImageNet	AlexNet, ResNet-18	0.6%, 2%
	INQ [210]	5-bit	32-bit	ImageNet	AlexNet, ResNet-50	−0.2%, 0.24%
	DLAC [177]	Binary	32-bit	ImageNet	ResNet-50	1.2%
	FFN [180]	Ternary	32-bit	ImageNet	AlexNet, VGG-16	−1.4%, −0.2%
Activation quantization	BNN [75]	Binary	Binary	ImageNet	AlexNet	29.8%
	XNOR-Networks [148]	Binary	Binary	ImageNet	AlexNet, ResNet-18	11%, 16%
	FGQ [123]	Ternary	8-bit	ImageNet	AlexNet, ResNet-101	3.65%, 4.39%
	DoReFa-Net [211]	Binary	2-bit	ImageNet	AlexNet	7.63%
	Training strategies [183]	Binary	2-bit	ImageNet	AlexNet	9.1%
	QNN [76]	Binary	2-bit	ImageNet	AlexNet	6.5%
	PACT [27]	3-bit	3-bit	ImageNet	ResNet-50	2.09%
	LQ-Net [199]	3-bit	3-bit	ImageNet	ResNet-50	3.19%
	HAWQ [38]	2-bit	4-bit	ImageNet	ResNet-50	1.91%
	HitNet [181]	Ternary	Ternary	PTB	LSTM	14 (PPB)
	Q-BERT [158]	2/3-bit	8-bit	SST-2	BERT	0.92%

framework to conduct sparse quantized training. Additionally, logarithmic data representation [126] was introduced to explore low precision quantization. Incremental Network Quantization (INQ) [210] only quantized half of the weights in each step, which leads to unnoticeable accuracy loss. Soulie et al. [163] introduced a compression method in the fully connected layers during the learning phase, which adds a regularization term to the loss function to make weights centered on +1 or −1.

Binary quantization takes linear quantization methods to the extreme, which constrains the parameters' only two values (e.g., 1 and −1). Although binary networks achieve up to 32× compression rate theoretically, binary networks reach an actual 10.3× compression rate [183]. BinaryConnect [31] implemented the training of the DNN with binary weights during the forward and backward propagations, but with full precision weights during the parameter update. Binary weight network (BWN) [148] extended the idea of BinaryConnect and introduced a scaling factor to achieve better approximation between the BWN and the original CNN. BWN outperformed BinaryConnect by large margins and made the first attempt to evaluate binary networks on large-scale datasets like ImageNet. Ganesh et al. [177] achieved high accuracy with binary weight networks while accelerating the execution time by aggressively skipping operations on zero-values. Based on the observation that training binary weight networks can be intrinsically regarded as a hashing problem, BWNH [73] trains binary weight networks via hashing, and an alternating optimization method is introduced to learn the hash codes instead of directly

learning binary weights. Since quantization function is non-differentiable, most binary/ternary quantization techniques approximate the gradients of weights by a *straight-through* estimator. Leng et al. [99] proposed to train binary/ternary networks via Alternating Direction Method of Multipliers (ADMM), which decouples the continuous parameters from the discrete constraints of networks and transforms the original hard problem into several subproblems.

Similar to binary weight networks in Reference [148], ternary weight network (TWN) [100] extended the idea of BWN and constrained all weights to be ternary values (e.g., +1, 0, −1) that can be presented by 2 bits. TWNs have stronger expressive abilities and are more effective than their binary counterparts. Based on the observation that untrainable and symmetrical ternary values may reduce the accuracy of ternary weight networks, Trained Ternary Quantization (TTQ) [212] learned both ternary values and ternary assignments during back-propagation, which could also be viewed as non-linear quantization. Combining weight matrix approximation with quantization, Fixed-point Factorized Network (FFN) [180] used fixed-point decomposition to quantize all weights as ternary values.

Linear Quantization of Activations. Activation quantization can reduce the requirement of storage and computation resources. Memory demand reduction comes from the decrease in model parameters and intermediate activations. Besides, the network can be executed more efficiently after activation quantization. Binarized Neural Networks (BNN) [75] was proposed as an extension to BinaryConnect [31], where both weights and activations were binarized, and 1-bit exclusive-not-or (XNOR) operations are used to drastically reduce memory usage and replace most multiplications. The bitwise neural network [89] achieved competitive performance by quantizing all inputs, weights, biases, hidden units, and outputs with 1-bit to dramatically save computation resources. XNOR-Networks [148] was presented as an extension to BWN and binarized both the inputs and weights. Otherwise, XNOR-Networks leveraged the binary operations to approximate convolution operations, achieving up to 58× speed up in CPUs on ImageNet. Fine-Grained Quantization (FGQ) method [123] not only ternarized the weights but also constrained the activations to 8-bit or 4-bit. DoReFa-Net [211] and QNN [76] were proposed to train CNNs with low bit-width weights and activations using low bit-width gradients method. Tang et al. [183] empirically proposed several effective training strategies for binary networks for higher accuracy and higher compression rates. Parameterized clipping activation function (PACT) [27] was presented to replace ReLu, and a more appropriate quantization scale was obtained by using the trainable clipping parameter. Unlike previous works that used hand-crafted and fixed quantization schemes, LQ-Nets [199] jointly trained quantizers and the quantized DNN. Based on the observation that the sensitivity of DNN layers to quantization was different, HAWQ [38] utilized Hessian spectrum to automatically find the right quantization precision of each layer.

Despite its impressive success in CNNs, quantization still suffers from significant accuracy degradation in RNNs. Based on the observation that the weights and activations had different distributions, HitNet, a hybrid quantization method, was presented in Reference [181]. HitNet is a hybrid ternary RNN that applied a different quantization method to weights and activations. A sloping factor was introduced into activation functions to reshape the distribution of activations. Shen et al. [158] extended HAWQ [38] on BERT and presented a new group-wise quantization scheme to bridge the accuracy gap between BERT and Q-BERT.

3.2.2 Non-linear Quantization. The key idea of non-linear quantization is to divide weights into a few groups, and each group shares a single weight. Non-linear quantization needs to perform classification and clustering, which is not hardware-friendly. Many approaches have been proposed to reduce weight classes. Gong et al. [49] addressed the model storage issue by vector quantization, which applied k-means clustering method to the weights. HashedNets [20] randomly

Table 5. A Comparison of Different Model Distillation Techniques

Method	Dataset	Teacher network	Teacher acc	Student network	Student acc
Softmax [9]	CIFAR-10	4-layer CNN	88.0%	2-layer CNN	85.8%
Knowledge distillation [69]	CIFAR-10	ResNet-26	94.55%	ResNet-14	88.21%
Noisy teacher [157]	MNIST	LeNet variation	99.03%	3-layer network	99.14%
FitNets [151]	CIFAR-10	5-layer Maxout (9M)	90.18%	19-layer Maxout (2.5M)	91.63%
Attention transfer [195]	CIFAR-10	NIN-wide (1M), ResNet-26	94.72%, 94.55%	NIN-thin (0.2M), ResNet-14	91.67%, 88.84%
FSP [194]	CIFAR-10	ResNet-26	92.55%	ResNet-14	89.92%
Adversarial example [68]	CIFAR-10	ResNet-26	92.55%	ResNet-14	90.34%
BERT-PKD [167]	SST-2	12-layer BERT	93.5%	3-layer BERT	87.5%
Transformer distillation [82]	SST-2	BERT (109M)	93.5%	BERT (14.5M)	92.6%
Optimal subwords [207]	SST-2	BERT (109M)	93.5%	BERT (19M)	88.4%
Distilled BiLSTM [172]	SST-2	BERT (109M)	93.5%	BERT (10.1 M)	90.7%

assigned weights into hash buckets based on a simple hash function, and each hash bucket shared a single weight. Quantized CNN [185] quantized both fully connected layers and convolutional layers and minimized the estimation error of the response of each layer.

3.3 Model Distillation

Model distillation is different from methods mentioned above. It utilizes dark knowledge transferred from a larger teacher network to train a smaller student network. Therefore, model distillation only supports training from scratch. By imitating the teacher network, the student network outperforms the model that is trained merely through class labels. The key to model distillation is to define the transferred knowledge. As shown in Table 5, we briefly summarize the performance of different model distillation methods.

To compress models based on knowledge transfer was first proposed in Reference [14]. A compressed model is trained with pseudo-data labeled by the original network. However, this method cannot be applied to deep models. To address this issue, in Reference [9], a compressed model mimicking the output distributions of the deep network is developed by shifting knowledge from a pre-trained teacher network to a compressed student network via softmax.

Hinton et al. [69] introduced the knowledge distillation (KD) method to ease the training of knowledge transfer, where the high temperature was introduced into softmax to soften the teacher's output during the training phrase. To learn from multiple teachers, Sau et al. [157] proposed a simple methodology that introduced a noise-based regularizer to simulate this effect, which led to a considerable improvement in the performance of the student network. Romero et al. [151] proposed the FitNets to train a thinner and deeper student model. The student model achieves a higher accuracy by taking advantage of the deeper network architecture. Besides, hint-based training, which leveraged the intermediate representations of the teacher as a hint to train the student network, was also devised. To better learn the intermediate representations of the teacher, Sergey et al. [195] proposed Attention Transfer (AT) to train the student network by mimicking the attention maps of a teacher network to greatly enhance the performance of the student network. The use of both activation-based and gradient-based spatial attention maps is proposed.

The essence of model distillation is to define the distilled knowledge. The distilled knowledge is represented as the form of an FSP matrix in Reference [194], generated by the features from two

Table 6. A Comparison of Different Designed Compact Network

CNN architecture	Top-1 ImageNet Accuracy	Number of parameters (Million)	Computation complexity (MFLOPs)	GPU[1] time (ms)	ARM[2] time (ms)
SqueezeNet	57.2%	1.2	-	-	-
MobileNet	70.6%	4.2	569	0.51	154
MobileNetV2	72.0%	3.4	300	0.69	112
ShuffleNet	71.5%	3.4	292	0.76	97
ShuffleNetV2	72.6%	3.5	299	0.49	85
ChannelNet	70.5%	3.7	407	-	-
IGCV2	70.7%	2.2	564	1.54	204
IGCV3	72.2%	3.5	318	1.22	159

[1]Single NVIDIA GeForce GTX 1080Ti.
[2]Qualcomm Snapdragon 810.

layers. Applying distillation methods for multi-class object detection is more challenging image classification, first successfully addressed in Reference [16]. With weighted cross-entropy loss to resolve the imbalance between the background class and the object class, a teacher bounded loss is designed for the regression problem [151] to better utilize intermediate teacher distributions. Mobiledeeppill [196] used multiple network design strategies to develop student networks, which combines the network design with knowledge distillation. Heo et al. [68] leveraged the information related to the decision boundary, namely, adversarial examples, to enhance the performance of knowledge distillation.

Patient knowledge distillation [167] was devised to allow the lightweight student network to patiently learn from multiple layers of the teacher BERT. A Transformer distillation method [82] was presented to compress the computation-expensive BERT to TinyBERT through multi-level distillation processes, such as embedding-layer distillation, attention-based distillation, hidden states–based distillation, and prediction-layer distillation. To reduce the size of the student model, a significantly smaller vocabulary was introduced into the student model in which tokenizations were incompatible with the teacher model for the same sequence [207]. Knowledge distillation from BERT into single-layer BiLSTM for specific tasks was also investigated in Reference [172].

3.4 Network Design Strategies

The massive design space of neural networks can be exploited for compression. It is promising to design low-cost and efficient network architecture. We give a summary of existing network design strategies for compression, as shown in Table 6. The missing values in the table are due to a lack of data from the referenced paper on the corresponding model.

The global average pooling was first proposed in Network in Network (NIN) [105], with the fully connected layer removed to reduce parameters of CNN models. NIN utilized 1×1 convolution to reduce the computation. It was proved that these strategies are effective in reducing storage and computation requirements, and they had been widely adopted in many later well-known works, such as GoogLeNet [169], ResNet [64], and SquezzeNet [79]. Branching, referring to multiple group convolution, was extensively explored in the GoogLeNet and ResNeXt and proved to be a successful strategy. Three main strategies were employed in designing SqueezeNet [79]: replacing 3×3 filters with 1×1 filters, decreasing the number of input channels to 3×3 filters, and downsampling late to obtain convolutional layers with large activation maps. According to these strategies, the fire module, which consists of 1×1 squeeze filters, 1×1 expand filters, and 3×3 expand filters, was

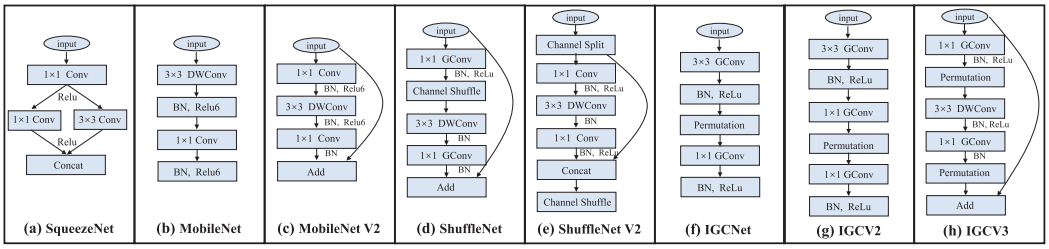


Fig. 6. The illustration of the building block in (a) SqueezeNet, (b) MobileNet, (c) MobileNet V2, (d) ShuffleNet, (e) ShuffleNet V2, (e) IGCNet, (f) IGCV2, (g) IGCV3.

introduced as the new building block of compact networks. The size of SqueezeNet is smaller than 0.5 MB with comparable accuracy by further using deep compression [61], which is 510× smaller than 32-bit AlexNet.

A mobile pill image recognition system named MobileDeepPill [196] is designed with five strategies. Compared with SqueezeNet, MobileDeepPill added two new strategies: splitting one layer into multiple layers and reducing the number of filters without splitting. Furthermore, the knowledge distillation-based compression method is used, achieving 5× reduction of the multi-CNNs model and 2× speedup in the inference phase.

As shown in Figure 6, many compact and low-cost network architectures have been designed. The depth-wise separable convolutions method was introduced in MobileNet [71]. Depth-wise separable convolution splits standard convolution into two layers: depth-wise convolutions and point-wise convolutions. The Depth-wise separable convolutions is used in Xception [28], an extension to Inception [169] and ResNet. Furthermore, combining ResNet with MobileNet, MobileNetV2 [154] introduced the shortcut connections to building blocks except for downsampling blocks and the inverted residual structure where input channels were first expanded to better leverage lightweight depth-wise convolutions. Besides, the linear bottleneck was introduced in the narrow layer to avoid the ReLu function's damage to channels.

ShuffleNet [186] utilized point-wise group convolution to further reduce the computation cost of MobileNet and utilized channel shuffle operation to enhance the information flow across channels among groups through random permutation. Lyu et al. [116] introduced AutoShuffleNet, aimed to learn permutation matrices automatically in the training phase rather than to design by hand. Ma et al. [117] argued that the effect of methods should be evaluated by direct metrics or targeted platforms, not by indirect metrics (i.e., FLOPs). Furthermore, four practical guidelines were presented, which discouraged excessive group convolution, network fragmentation, and element-wise operations. ShuffleNetV2 was proposed according to these guidelines. Gao et al. [43] compressed the fully connected layer by replacing it with the convolutional classification layer. Moreover, channel-wise convolution was proposed for sufficient information fusion of feature maps in different groups.

Meanwhile, Zhang et al. [203] proposed IGCNet to develop group convolutions that add a secondary point-wise group convolution and permutation matrix. To reduce redundancy of dense matrices in IGCNet, IGCV2 [187] was proposed, which further decomposed the group convolution in the IGCNets' manner through theoretical analysis. In other words, a point-wise group convolution in IGCNets was decomposed into multiple point-wise group convolutions in IGCV2. The architecture of IGCV3 [166] is similar to MobileNetV2's, but it used point-wise group convolution instead of point-wise convolution. Singh et al. [161] proposed a novel convolution architecture, named HetConv, which, unlike standard convolution, utilized heterogeneous kernels to achieve 3× to 8× speedup.

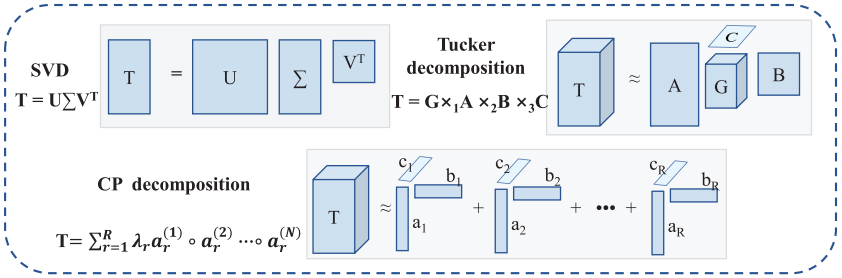


Fig. 7. Three tensor decomposition methods: SVD, Tensor Decomposition, and CP Decomposition.

3.5 Low-rank Factorization

The low-rank factorization technique has long been used to accelerate and compress neural networks. The key idea of low-rank factorization is to find an approximate low-rank tensor $\widehat{W}$, which is close to W and is easy to be decomposed. The convolution kernel $W^{X\times Y\times M\times N}$ can be viewed as a 4D tensor. The four dimensions X, Y, M, and N correspond to the kernel width, kernel height, and the number of input and output channels, respectively. Tensor decomposition approaches, such as CP decomposition and Tucker decomposition, can be used in convolution kernels. Low-rank factorization is helpful for both tensors and matrixes. We classify and summarize low-rank factorization methods based on the decomposition approaches, as shown in Figure 7.

3.5.1 Single Vector Decomposition (SVD) & Two-component Decomposition. Single vector decomposition (SVD) is the most popular matrix decomposition method that can be used in both the convolutional layers and the fully connected layers. By merging three of the four dimensions, the kernel becomes a 2-D matrix. There are mainly two ways for dimension reduction: merging the dimensions X, Y, and M, and merging the dimensions X, Y, and N. The former is introduced in Reference [204], and the latter is symmetric with the former.

Denton et al. [35] derived approximations of a lower complexity by using the redundancy inherent in the convolutional filters. By using bi-clustering, outer product decomposition, and SVD, the method achieves 2× acceleration for a convolutional layer with negligible loss in accuracy. SparseSep [12] is a sparse coding-based factorization to optimize the inference phase. The convolution kernel separation method is utilized to minimize the overall computation cost based on two-component decomposition and SVD. Cheng et al. [170] demonstrated that two-component low-rank decomposition has an exact global closed-form and data-independent approximation, which is more effective than iterative methods. Moreover, low-rank constrained CNNs are trained from scratch using the batch normalization. By decomposing the large vocabulary embedding matrix into two small matrices, the parameters in BERT can be significantly reduced [93].

3.5.2 CP Decomposition. The Canonical Polyadic Decomposition (CPD), an extension of SVD, has found application in statistics, signal processing, psychometrics, linguistics, and chemometrics. A CP-decomposition acceleration method is proposed in Reference [98], with four convolutions of size 1×1, X×1, 1×Y, and 1×1. It is hard to get the best low-rank approximation in the CP decomposition for the reason that it may not exist in general cases. To overcome this difficulty, non-linear least squares are leveraged to calculate the CP decomposition.

3.5.3 Tucker Decomposition. Tucker decomposition decomposes a tensor into a set of matrices corresponding to different core scalings along with each mode and one small core tensor. Therefore, the Tucker decomposition can be seen as a higher-order PCA.

Zhang et al. [201] introduced a Tucker decomposition model and a back-propagation strategy–based learning algorithm that is orchestrated to train the model to compress weight tensors in the fully connected layers. A one-shot whole network compression [90] is proposed, including three stages: rank selection, Tucker decomposition, and fine-tuning. Tucker-2 decomposition, known as Generalized Low Rank Approximations of Matrices (GLRAM) [193], is performed in the convolutional layers, and Tucker-1 decomposition (i.e., SVD) is used in the fully connected layers.

3.6 Other Techniques

Apart from the above five categories of mainstream techniques, there is a scattering of other techniques for network compression and acceleration.

Cong et al. [29] converted the computation in the convolutional layers of a CNN to Convolutional Matrix Multiplication and used the Strassen algorithm, which uses more additions to reduce the number of multiplications, to reduce the computation by up to 47%. A direct sparse convolution algorithm is proposed in Reference [140] that can efficiently convert sparse convolution operation into dense matrix multiplication. Escort [21] developed the direct sparse convolution approach to efficiently run on GPUs, which orchestrates the parallelism and adds a sum buffer to maximize data locality. Prabhavalkar et al. [145] used a shared recurrent projection matrix to compress LSTM RNN model by jointly decomposing inter-layer and recurrent matrices. Deep Fried Convnets [192] uses Adaptive Fastfood transform to compress the fully connected layers with negligible accuracy loss on the MNIST and ImageNet datasets. However, Deep Fried Convnets are not greatly fast, because the computation in CNN is dominated by the convolutions. Li et al. [103] proposed LightRNN, which used the 2-component shared embedding for word representations to reduce the model size and running time of RNNs for natural language processing tasks. To reduce energy consumption, Sarwar et al. [156] replaced some weight kernels of a CNN with Gabor filters in the first and second layers. Half-fixed/half-trainable configuration is adopted to best balance the accuracy and training complexity. To reduce the training time, Sainath et al. [153] parallelized the gradient computation during cross-entropy and sequence training and used a low-rank matrix factorization to compress the network. Kim et al. [88] designed a lightweight network for object recognition by using C.RELU and inception structure. According to the convolution theorem, Fast Fourier Transformation (FFT) can significantly reduce the convolution computation cost of the large kernel. Another efficient method [96] using Winograd's minimal filtering algorithms (WMFA) has a great advantage in the small convolution kernel, and its key idea is to reduce the number of multiplications.

Due to limited memory on chip, outputs of hidden layers cannot be stored in the memory on chip or SRAM once. Furthermore, because of low memory bandwidth, the data flow from memory on chip to memory off chip is time-consuming and dramatically slows the speed of computation. To deal with these issues, new data flow methods are proposed. Alwani et al. [4] presented a fused-layer CNN accelerator to reuse inter-layer intermediate data on chip by fusing the computation of multiple CNN layers through the pyramid-shaped multilayer sliding window.

3.7 Hybrid Techniques

Some studies have attempted to integrate these orthogonal techniques to achieve more significant performance. Han et al. [61] designed a "deep compression" pipeline that included pruning and quantization techniques as well as Huffman encoding, working together to get 35× to 49× reduction with the same accuracy, as shown in Figure 8. Similarly, Tung et al. [174] proposed CLIP-Q, which combined pruning technique with weight quantization technique jointly instead of sequentially. CLIP-Q obtains 51× reduction for AlexNet.

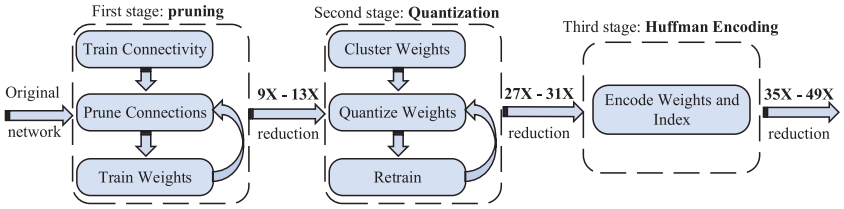


Fig. 8. The "deep compression" pipeline [61]: pruning, quantization techniques, and Huffman encoding.

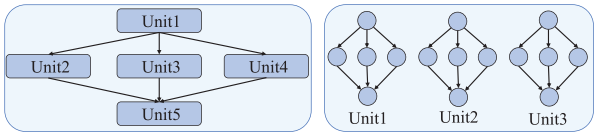


Fig. 9. Two modes to design parallelism: model parallelism (left) vs. data parallelism (right).

Goetschalckx et al. [47] jointly exploited SVD, retraining, pruning, and clustering to achieve superior compression in neural networks, outperforming prior art with 5× increased compression capabilities without accuracy loss. Mobiledeeppill [196] used multiple network design strategies to build student networks and combined the network design with knowledge distillation. Besides, the group convolution and point-wise convolution strategies, similar to that used in IGCNet [203], are leveraged to cheapen the student network for model distillation [32]. The combination of quantization and model distillation was also investigated in Reference [142].

4 HARDWARE

The applications of deep learning models on mobile devices pose great demand for various hardware resources. The hardware resources of traditional mobile devices are unable to meet the computational requirements of deep learning models. Therefore, researchers have been dedicated to improving various hardware to enable deep learning tasks to be performed on the mobile end.

4.1 Hardware Solutions

The amount of computation power that deep learning models require is huge, while the resources of mobile and embedded devices are limited, so we need to choose hardware solutions that can tackle the problems of energy consumption, computation complexity, and cost. First, to reduce energy consumption, we can design special dataflow architectures to coordinate the computations. Other techniques such as new memory architectures, processing in memory, and processing in sensor can also be used to reduce energy consumption. Second, to reduce the cost, we can choose cheaper solutions such as FPGA accelerators. In addition, the mass production of ASIC chips is also an economical choice. Third, some methods can be used to accelerate computation, such as designing compressed algorithms and improving parallelism. Three main solutions have been presented and analyzed in the following contexts, namely, parallelism, data flow architecture, and processing in sensor.

4.1.1 Parallelism. It is fundamental to properly allocate hardware resources to make the calculations fully parallel. Based on the distributed parallelism, we can divide the hardware accelerator parallel methods into model parallelism and data parallelism, as shown in Figure 9.

- *Model parallelism.* Model parallelism is also spatial parallelism. Each computation unit is responsible for the computation tasks of different parts of the model but uses the same set of data. For example, different output feature maps in a layer can be calculated using the

same input feature maps and different kernels at the same time. Since output feature maps are independent of each other, they can be calculated simultaneously.

- *Data parallelism.* Data parallelism divides and distributes computing tasks on subsets of data. Different computing units have the same copies of the model but are assigned different subsets of data, and all the calculation results are combined to produce the final results. In other words, multiple computing units perform the same operation at the same time but use different inputs. For example, in a convolutional neural network, the convolution operations consist of many multiplication and accumulation processes, which can be assigned to several multiply-accumulate units so some subsets of pixels can be processed simultaneously as long as they are independent of each other.

4.1.2 Dataflow Architecture Design. We can augment the degree of parallelism by adding hardware resources, but this will increase the pressure of memory reads and writes. A slow memory rate will become a bottleneck for calculations. Reading and writing memory repeatedly will also raise energy consumption, which is a critical problem for mobile devices. Many methods have been presented to address the memory bottleneck, such as adding buffers, data reuse (filters, pixels of input feature maps), using locality principles, and reducing data precision. Existing accelerators for dataflow can be grouped into four categories: weight stationary, output stationary, no local stationary, and row stationary [23].

4.1.3 Processing in Sensor. Besides processing in memory, processing directly in the sensor is also becoming a trend. Addition and multiplication operations in the analog domain at the sensor can be more energy-efficient. RedEye [104] tries to tackle the analog readout bottleneck by moving convolutional neural network processing into the analog domain of the image sensor. ASP Vision [17] uses Angle Sensitive Pixels (ASPs) and bio-inspired CMOS image sensors instead of traditional image sensors. ASPs are used to compute the first convolutional layer, which leads to a 90% sensor power consumption reduction and 90% bandwidth reduction compared to traditional deep neural network pipelines.

Hardware resources are limited on mobile devices. Instead of blindly pursuing the speed of operation, it is better to achieve a balance of production costs, energy consumption, and other factors. In the rest of this section, we will introduce the deployment of deep learning models regarding various hardware on mobile devices, including GPU, FPGAs, and ASIC chips.

4.2 Mobile GPU

There are many differences between mobile GPUs and desktop GPUs due to limited space and power. Mobile GPUs have limited memory and fewer GPU cores. The low memory bandwidth is also a bottleneck of mobile GPUs. Many mobile deep CNN libraries only support multi-core mobile CPUs, such as Caffe Mobile and Torch Mobile. Moreover, many commonly used CUDA-based libraries do not support mobile GPUs. Instead, RenderScript and OpenCL are widely employed in mobile and embedded devices. OpenCL supports a wide range of commodity mobile GPUs compared with CUDA-based devices. RenderScript is most commonly used as the Android's parallel computing platform. Different from desktop GPUs, thread synchronization is not available in RenderScript.

Due to the differences between desktop GPU and mobile GPU, developers require to convert models trained in desktop GPUs into the models suitable for mobile GPUs through model converter and model loader. The model converter adjusts the parameters and configurations of a trained model to produce a new model ideal for mobile GPUs. The model loader then loads the new model on mobile deep learning frameworks and allocates memory spaces. Since the memory space is small but the input data are often large, all weights of CNN models are stored in the device memory.

Table 7. A Comparison of Different GPU-based Methods

Method	Platform	Devices	Model	CPU time(ms)	GPU time(ms)	Acc/Acc drop(%)	Enery (mJ)
DeepSense [77]	OpenCL	Samsung Galaxy Note 4	VGG-F	9177	259	76.3/4.6	665
CNNdroid [95]	RenderScript	Samsung Galaxy Note 4	AlexNet	20767	482	80.8/0	400
MobileConvNet [133]	RenderScript	LG Nexus5	SqueezeNet	42933	141	80.3/0	106
MobileRNN [15]	RenderScript	LG Nexus5	LSTM	142	29	-/0	-

Huynh et al. [77] proposed a DeepSense framework built on OpenCL. DeepSense can be used in various GPU architectures such as Adreno and Mali. In DeepSense framework, the CPU and mobile GPU work together to run the model, where the mobile GPU takes on heavy computation tasks such as convolutional layers and the data representation. Some optimization strategies such as memory vectorization, half floating-point, and branch divergence elimination are adopted to further reduce latency. DeepMon framework [78] is built on OpenCL and executes large DNNs to extract features from continuous video frames. DeepMon adopts Tucker-2 to decompose matrices in convolution layers to speed up the multiplications between tensors. DeepMon also adopts the optimization strategies, such as half floating-point precision and memory representation. CNNdroid [95], a GPU-accelerated library for Android devices, was specifically designed to execute existing desktop models on Android devices. To reduce the overhead of loading corresponding matrices from the SD card into RAM, certain layers are chosen to be kept in RAM. In desktop GPUs, convolution operations are usually unrolled and converted to matrix multiplication, which requires a considerable amount of memory that is not affordable for mobile devices with small cache and memory sizes. Therefore, the matrix multiplication is transformed to dot product that can be executed on mobile GPUs more efficiently. Motamedi et al. [133] proposed efficient parallelization techniques of CNN inference on mobile GPUs. The number of feature maps is computed independently by using one thread to calculate one output value. To facilitate data read, the data are reordered to be stored in a layer major format. A trade-off between the numbers of threads and the amount of the code that should be executed in each thread is made due to limited resources. We compare the performance of these frameworks in Table 7. We can observe that DeepSense sacrifices accuracy to reach lower latency, but the energy cost is high. CNNdroid is specially designed for Android devices, which achieves remarkable speedup and energy saving on AlexNet, but the CPU time is relatively high. Besides, lightweight networks, such as SqueezeNet, may be used to further reduce inference time and energy consumption on mobile GPUs [133]. MobiRNN [15], a mobile GPU offload optimization framework, was specially designed to accelerate RNN models that cannot be handled by DeepSense or CNNdroid. MobileRNN has a light overhead by using mobile parallelization to pack vector products into fewer work units.

Besides, ARM compute library [8] was specially devised to accelerate many tasks, including image processing, computer vision, and machine learning, for ARM Mali GPUs. The open-sourced CLBlast project [136] was launched to provide computational acceleration in OpenCL/CUDA hardware including embedded and low-power GPUs. The hand-tuned OpenCL outperformed CLBlast as reported in Reference [19] due to the small size of the input image.

4.3 FPGA

Traditionally, there are two options for hardware computing: general-purpose processors and customized chips. General-purpose processors are not very flexible, and customized chips are costly.

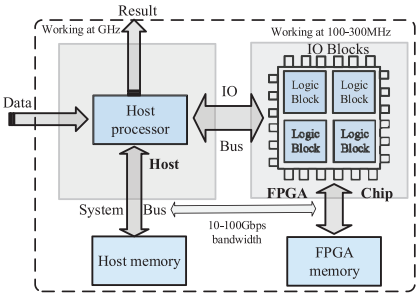


Fig. 10. A typical structure of an FPGA-based neural network accelerator.

Field-Programmable Gate Array (FPGA) is a semi-customized circuit, which is less expensive than customized chips while overcoming the shortcomings of a limited number of gate circuits on general-purpose processes.

FPGA is implemented using a Hardware Description Language (HDL). The execution of the instructions in HDL is not in order. In other words, the running code is not executed sequentially. Instead, the logic gate structure is generated after reading the code. These logic gate structures operate in parallel, generating output at the same time as inputs provided, so FPGA is mainly based on pipeline-parallel computing. Unlike PCs or single-chip microcomputers, FPGA does not use von Neumann architecture or Harvard architecture, which stores instructions and data in memory first and then fetches them when needed. Hence, an FPGA can execute instructions simultaneously rather than serially like embedding multiple CPUs in a single chip. In general, the functions that the CPU can implement can be accomplished by FPGA in a hardware design method. In addition, the FPGA architecture can be tailored for an application, such as mobile DNNs [92]. Moreover, FPGA consumes less power than CPU and GPU given the same task. Therefore, FPGA is ideal for accelerating mobile deep learning models.

Nevertheless, to efficiently leverage FPGAs for model deployment, developers need to master the programming language Verilog or VHDL and acquire the knowledge of digital design and circuits. In addition, compilation synthesis, placement, and routing may take a long time, e.g., a large project may be executed in several hours. There are many High-Level Synthesis (HLS) tools to assist programmers in generating low-level Register-Transfer Level (RTL) or HDL code [92], model-based frameworks, C-based frameworks, high-level language-based frameworks, HDL-like languages, and parallel computing frameworks [10].

As shown in Figure 10, FPGA architecture is often composed of three parts: I/O pads, logic blocks, and routing channels. For deep learning models, a single FPGA chip is not able to accomplish all computation tasks, so the FPGA is usually working with the host processor and off-chip memory. Global buffer can be used to hide DRAM access latency and exploit data reuse. The host processor writes data and instructions to the memory. The FPGA reads from the memory over IO Bus, performs computation, and finally writes back the results.

Table 8 presents the performance comparison of existing FPGA-based deep learning frameworks. Motamedi et al. [134] designed an architecture template that exploits all resources of parallelism in a DCNN to utilize the optimal combination of available parallelism strategies and achieve the best performance using the least time with the limited sources of the target FPGA chip. Gokhale et al. [48] presented the nn-X system, which utilizes large parallelism to perform convolutions and takes full advantage of hardware resources. The system adopts data-streaming architecture to run the required operations during every clock. When running neural networks, nn-X could reach 227 GOPS while consuming less than 4 watts of power. Zhang et al. [197] introduced a

Table 8. Comparison of Different FPGA-based Deep Learning Frameworks

	ASPDAC2016 [134]	nn-X [48]	FPGA2015 [197]	FPGA2016 [146]	FPGA2016 [165]	PLACID [135]	
Data representation	32-bit float	16-bit fixed	32-bit float	16-bit fixed	8-16 -bit fixed	16-bits fixed	16-bits fixed
Frequency	100 MHz	150 MHz	100 MHz	150 MHz	120 MHz	100 MHz	100 MHz
FPGA Chip type	VX485T	Zynq XC7Z045	VX485T	Zynq XC7Z045	Stratix-V(GSD8)	Cyclone	VX485T
CNN computation	1.33 GOP	0.55 GOP	1.33 GOP	30.0 GOP	30.9 GOP	1.33 GOP	1.33 GOP
Number of DSPs	2,800	900	2,800	900	1,963	87	2,800
Number of slices	75,900	-	75,900	-	-	-	75,900
Power	-	8 W	18.61 W	9.93 W	25.8 W	-	-
Performance	84.2 GOPS	23.2 GOPS	61.6 GOPS	14.2 GOPS	136.5 GOPS	12.1 GOPS	445 GOPS

roofline-model-based method to optimize FPGA-based accelerator design. The computation of convolutional layers is minimized by loop unrolling, loop pipelining, and tile size selection. Local memory promotion and loop transformations are adopted for data reuse, and CTC ratio is used to optimize memory access. Qiu et al. [146] analyzed the current CNN models and found that the convolution operation is the most computationally intensive and the fully connected layer is very sensitive to memory. Based on these observations, the dynamic precision data quantization flow and SVD technique are utilized to optimize the memory access with no accuracy loss. Suda et al. [165] presented an accelerator using OpenCL framework. Since convolutions are computation-intensive and performance-critical, a scalable convolution module is developed, which maps 3D convolution operations as matrix multiplications. Motamedi et al. [135] introduced PLACID, a platform that could create accelerators for DCNN automatically. Similarly, PLACID leverages design space exploration to balance different parallelism sources and improve the performance, which can generate suitable accelerators for given DCNNs and FPGA platforms using only a few hours.

4.4 ASIC

An ASIC is an integrated circuit designed and manufactured to meet specific user requirements and the needs of a specific electronic system, thus can be used to develop dedicated deep learning acceleration chips. ASICs have a smaller size, lower power consumption, improved reliability, improved performance, enhanced confidentiality, and reduced cost when compared to general-purpose integrated circuits in mass production.

As shown in Figure 11, FPGA and ASIC are different in several ways. On the one hand, ASIC is customized chips with complex design flow and implementation processes, leading to a relatively high cost. In comparison, FPGA is semi-customized chips, whose design flow is simpler and takes a shorter time. Due to the difference in costs, ASIC chips are suitable for mass production and large-scale projects, while FPGA chips can be used for small-scale production and quick turnover rate. On the other hand, to support programmable flexibility, it is inevitable for FPGA to have unnecessary elements and increased chip size. In contrast, the chip area of ASIC can be greatly reduced by a certain design and layout for a specific task. And FPGA consumes more energy than ASIC under the same processing conditions. The operation circuit of FPGA is based on a lookup table, which is energy-consuming. ASIC can redesign the chip architecture and reduce power consumption. Accordingly, the speed of FPGA is generally slower than using an ASIC.

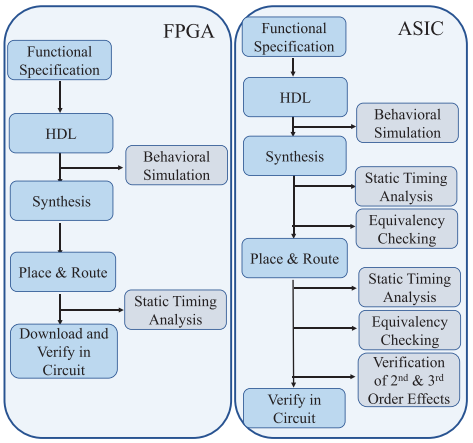


Fig. 11. The design cycles of FPGA and ASIC chips [188].

TrueNorth is a kind of Neural Processing Unit (NPU). Deep convolutional neural networks are implemented on TrueNorth chips [40] with 4,096 cores (each having 256 inputs and 256 neurons) and a 256×256 synaptic array. The network on the chip has neuron parameters, synapses, and programmable connectivity. Eyeriss [23] is a chip with a spatial architecture for convolutional neural networks. A dataflow named Row-Stationary (RS) is designed to maximize input data reuse and spatial parallelism and minimize partial sum accumulation cost. RS dataflow can be used to optimize various kinds of energy costs due to data movement in different conditions. Myriad 2 [131] is a mobile Vision Processing Unit (VPU) with computational imaging hardware accelerators, 12 vector processors, RISC-RT, RISC-RTOS, and memory fabric. RISC processors are used as controllers. The vector processors can run complex algorithms in parallel at high performance and low power. By utilizing data locality, the memory is optimized for low power, zero latency, and sustained high performance. Recently, Intel has proposed Myriad X [80] with 16 powerful processors and an ultra-high throughput intelligent memory fabric, making its performance 10× better than the performance of Myriad 2. DianNao [18] is an accelerator that focuses on memory usage. Its storage is divided into different parts to attain a design balance between power consumption and performance. DianNao makes the most use of data locality to minimize memory transfers. Inspired by DianNao [18], Du et al. [39] presented ShiDianNao, an accelerator for CNN, which is fast and energy-efficient enough and more suitable for embedded systems. Leveraging its weight-sharing feature, an entire CNN model is loaded into SRAM to reduce the memory overhead caused by accessing the DRAM. ShiDianNao is 60× more energy-efficient than DianNao. EIE [60] is an efficient inference engine that can run DNN compressed by deep compression, where pruning and weight-sharing techniques are used. EIE fully exploits sparsity in activations and weights in fully connected layers with compressed sparse column representations. The model can be fetched from SRAM instead of DRAM. EIE greatly reduces energy consumption compared with CPU and GPUs. SCNN [139] introduced sparse planar-tiled input-stationary Cartesian product dataflow, which maximally reuses data in a compressed encoding to reduce the memory access to input feature maps. Similar to EIE, SCNN also considered the sparsity of activations and weights. Cambricon X [202] introduced a hardware module called IM (Index Module) in the buffer controller to select needed neurons for PEs. Cambricon X can run both dense and sparse neural networks and has better performance and energy efficiency than DianNao.

Also, some boards and embedded systems are available in the market that support deep learning, e.g., Hikey 970 [1], Jetson Nano board [137], Edge TPU (Tensor Processing Unit) Devices Coral [50], and EdgeBoard [11]. Hikey 970 [1] is an AI platform with a powerful Neural Process Unit. It is equipped with HiAI SDK, which can provide AI foundation, engine, and service. Jetson Nano board [137] has Quad-core 64-bit ARM CPU and 128 NVIDIA CUDA cores and can run several neural networks simultaneously in an energy-efficient way. It is equipped with Jetpack SDK, which supports TensorRT, cuDNN, GPU accelerated application, and some other libraries for computer vision applications. TPU [83] is a chip designed specifically to accelerate the computing of deep neural networks. It has a large on-chip memory and can perform 8-bit integer arithmetic. Different from TPU that is powerful enough to perform training, edge TPU only conducts inference on the trained model. Coral products [50] have an edge TPU as a coprocessor and supports tensorflow lite. The edge TPU can perform 4 trillion operations per second and costs only 0.5 watts for each TOPS.

A comparison of the performance of different deep learning frameworks on different edge devices (including mobile devices, FPGA, and ASIC) was conducted in Reference [59], based on which the optimal framework can be determined for different tasks on different devices.

4.5 Memory

The performance of the memory will directly affect the performance of a processor. Deep learning models have a high memory demand. Mobile devices have limited memory, and reading and writing memory consume a lot of power. Deploying deep learning models on mobile devices requires not only fast memory read and store but also less energy consumption. Processing data directly in memory can reduce data movement. Zhang et al. [200] proposed to perform multiplication and addition in-place by the bit cells of an SRAM to avoid energy-intensive accesses. Crossbar resistive memory architectures based on memristors are proposed to perform dot-product operations in parallel, but may result in high overheads of analog-to-digital conversions and digital-to-analog conversions.

In addition, various new memory technologies have been proposed recently. Embedded DRAM (eDRAM) [86] is a dynamic random-access memory that can be integrated on the same die as the processor. eDRAM is often used on ASIC chips and microprocessors. Though costing more than traditional DRAM per bit, eDRAM has a much higher density and is more energy-efficient. eDRAM also allows for wide buses and high operation speeds. Hybrid Memory Cube (HMC) [81] is a 3D DRAM architecture that combines through-silicon vias and microbumps to connect several dies of memory cell arrays on top of each other. A logic die is used to perform functions such as data distribution, array repair functions, high-speed host interface, address/control, and refresh control. Compared with traditional DRAM, HMC has more data banks, resulting in a higher density and bandwidth. High Bandwidth Memory (HBM) [84] is similar to HMC, which consists of a logic die and DRAM dies, but the operating frequency and bandwidth of the two are often different. The performance of the above three memory technologies is better than traditional memory, but their price is much higher. eDRAM has been applied to some mobile devices, while HMC and HBM are only used on servers and supercomputers. Thanks to their high storage density, area-saving, and low energy consumption, these memory technologies may be used in mobile devices in the near future, which will further boost the development of deep learning models on mobile devices.

To improve memory performance, many deep learning accelerators [146, 197] use ping-pong technology to manage the buffer. As shown in Figure 12, in the first buffer cycle, the input dataflow is stored in the buffer module A. In the second buffer cycle, the input dataflow is stored in the buffer module B through the selection of the input switch. The data stored in the buffer module A at the first cycle will be sent to the operation processing unit through the selection of the output switch.

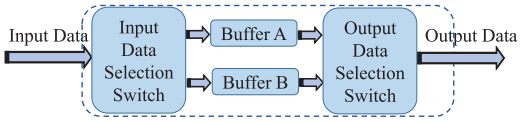


Fig. 12. Using ping-pong technology to manage buffer.

During the third buffer cycle, the input buffer and the output buffer are switched. The ping-pong structure can ensure the continuity of data input and output, as well as reduce the workload of the data processing module. The processing rate is only half of the data stream transmission rate. When running CNNs, the data between layers is dependent. The output feature maps of the prior layer are the input feature maps of the next layer. Therefore, the ping-pong technique can be used between the layers.

5 SOFTWARE FRAMEWORKS

Software frameworks provide ready-to-use base structures and building blocks that are important to implement deep learning models such that it is unnecessary to build the model from scratch. There are many available frameworks for desktop computers, such as Caffe, Theano, Tensorflow, and Torch. However, due to the limitation of battery power, memory, and computation resources, these frameworks are not suitable for mobile and embedded devices, e.g., the mobile GPUs can not match desktop GPUs in accelerating deep learning models. Although some software frameworks for desktop computers also include specific support for mobile devices, there are still two challenges: (1) the framework may be unavailable for mobile GPUs, and (2) the frameworks mainly accelerate the speed of training but not inference, which is critical for real-time applications. To address these problems, software frameworks specifically designed for mobile devices have been proposed.

DeepX [94] is the first software-based deep learning accelerator that takes an essential first step towards mobile deep learning. DeepX accelerates the deep learning inference by reducing the latency of fully connected layers. DeepX uses runtime layer compression and deep architecture decomposition to control the resources and optimize the computation, memory, and energy requirements across available hardware resources. DeepX splits computations into smaller and more manageable tasks that are allocated to different processors such as GPUs and CPUs, thus fully utilizing the mobile computing capability and achieving a balance between speed and resource consumption. However, DeepX relies on more powerful hardware. Boda-RTC [132], an open-source system, achieved competitive computational speed and greater portability by targeting the vendor-neutral OpenCL platform. Boda-RTC enables the rapid development of new computation kernels for the hardware targets and fast tuning of computation kernels for new hardware targets. DeepLearningKit [176] is a deep learning library and an open-source framework on Apple iOS. DeepLearningKit supports deep learning models pre-trained with popular frameworks such as Caffe, Torch, TensorFlow, Theano, and Pylearn. Sarkar et al. [155] produced OpenCL and RenderScript-based libraries for implementing DCNNs on mobile GPUs that are more suitable for mobile devices than CUDA-based frameworks.

There are many mobile software frameworks that are developed based on their desktop counterparts. RsTensorFlow [5], an extension to TensorFlow, can directly run models on Android devices trained with TensorFlow by leveraging RenderScript library. RsTensorFlow accelerates matrix multiplication by using RenderScript instead of the Eigen library. A RenderScript script kernel file is developed to parallelize the convolution operation. CaffePresso is a Caffe-compatible framework that can generate code and adapt implementation parameters automatically for various platforms

such as FPGAs, DSPs, GPUs, and multi-cores according to their different constraints. The specific strategies such as maximizing different levels of parallelism and optimizing data storage are used to reduce the time spent on performing convolutions. Besides, mobile deep learning frameworks have also received growing attention from the industries. TensorFlow lite [51] was released by Google in 2017 to enable TensorFlow model inference on mobile devices with a small binary size and low latency. And Caffe2 was presented by Facebook in 2017 to extend Caffe on mobile devices [41]. TensorFlow lite and Caffe2 have achieved comparable performance, which reached 175 ms and 158 ms, respectively, for SqueezeNet on Snapdragon 821 SoC.

6 APPLICATIONS

Due to page limitations, the context of applications of mobile deep learning is available as online supplementary materials.

7 DISCUSSION: PRIVACY ISSUES FOR MOBILE DEEP LEARNING

In this section, we discuss privacy issues for mobile deep learning. User data are private and may be sensitive, e.g., temporal locations, historical trajectories, and check-in time of social networks. By downloading the trained deep learning model locally on mobile devices for inferences, it seems that user privacy can be well protected. Nevertheless, due to the limited data and computing resources on mobile devices, the training process is still remotely conducted on the server. Therefore, the server may solicit mobile devices to upload their data for training or updating the deep learning models, which presents severe privacy risks [22]. How to protect sensitive data while achieving efficient on-device deep learning has received considerable attention in recent years.

Since the data are transmitted to the server for training, to encrypt the data is an intuitive solution to prevent information leakage. The data owner, i.e., the mobile device, sends encrypted data to two or three non-colluding servers. The servers run a secure Multi-Party Computation (MPC) protocol to train the model with encrypted data from one or multiple mobile devices. Later, mobile devices can download the model (which may be encrypted) for local inference. The computation overhead of mobile devices is light. Moreover, making inferences can also be outsourced to one or several servers by utilizing the homomorphic encryption techniques or MPC protocols. Several such frameworks have been designed [46, 85, 107, 128, 129, 150].

Although encryption provides strong security, the computation complexity on the server side is high. Instead, differential privacy techniques mask sensitive data more efficiently with random noises. Differentially private deep learning can be roughly divided into two classes. One is record-level privacy, which aims to protect every record in the training dataset. It can be implemented by perturbing the gradients [2] or the objective function on the client side [141]. The other is client-level privacy, which protects the existence of every client in the federated system [45, 122]. Noises are added to the summation of updates on the server side. Record-level privacy may incur a higher privacy cost, and accuracy loss, since noises are added to every update. Client-level privacy can achieve high efficiency only if there is a large number of users; otherwise, the privacy budget will be consumed rapidly. Apart from the above two cases, RONA [179] presents a framework of applying knowledge distillation to model compression with privacy guarantee in mobile applications. Sensitive data are only used to construct the teacher model that is not published, and all the outputs of the teacher model are perturbed by differential privacy.

Federated learning is a popular distributed learning paradigm proposed by Google to train a model collaboratively over millions of mobile devices [121, 206] while avoiding the breach of private data. In every round of training, mobile devices update a learning model locally and upload the parameters or gradients to a centralized server, then the server averages all updates and returns the result to devices to begin a new round. In this way, user data always stay within their devices.

Unfortunately, some recent works have shown that only sharing the parameters will cause privacy leakages [205]. An attacker can recover the training data from a trained model by exploiting the confidential information in classification [42], compromising the training code to encode the data into the model [162] or using Generative Adversarial Networks (GAN) to generate data with similar distribution [70, 182]. Furthermore, whether a data record exists in the training dataset or not can be inferred by constructing shadow models to simulate the behaviors of the target model [159]. To tackle these problems, some methods have been proposed to preserve the confidentiality of the parameters. Google designs a secure aggregation protocol to avoid revealing any individual's update to the server based on secret sharing [13]. Every update is masked by random vectors negotiated by multiple pairs of users. If half of the other users add the vector to the update and others subtract it from the update, the masks can be canceled when all updates are added, and the real value of the update can be protected with perfect secrecy. Moreover, utilizing mentioned client-level differential privacy techniques could prevent malicious users from inferring sensitive information about others.

8 FUTURE DIRECTION

The emerging trend to deploy deep learning models on mobile and embedded devices has attracted the attention of both academia and industry. However, we are still facing many challenges. In this section, we discuss some promising future directions in this field.

Adaptive/automatic compression. Current network compression and acceleration techniques involve many hyperparameters, such as the threshold of pruning and the bitwidth of linear quantization, which need to be determined based on empirical experiments and expert knowledge, which poses great pressure on mobile application developers. Additionally, it is critical to retrain the network by fine-tuning, which requires expert knowledge and considerable experiments. It is a promising direction to have adaptive/automatic compression and acceleration methods and tools, not relying on manual design, which can be readily used by developers. To the best of our knowledge, some explorations have begun [66].

Unsupervised compression. Labeled data are needed to retrain the network during compression to ensure accuracy maintenance. Labeled data may be unavailable in the real world sometimes, such as medical images. Moreover, unsupervised learning becomes more and more important, which benefits from the abundant unlabeled data in reality. Unsupervised compression techniques are desirable to address these difficulties.

Hardware technology development. Many new hardware technologies, such as HMC, HBM, and eDRAM, have been used on desktops and servers, but not on mobile devices due to their high cost. It is expected that these new memory architectures will be used on mobile devices in the near future. Some mobile chips are released but have not yet reached the scale of mass production. More efforts should be devoted to making a trade-off between price and performance. The application of these technologies on mobile devices should be promoted to boost the deployment of deep learning models on mobile devices. In addition, decomposition, pruning, and data quantization are often used to make a model hardware-friendly but may affect the performance. More hardware solutions should be proposed to solve the problem.

Memory access optimization. Memory access is very energy-consuming, and energy conservation is a key issue for mobile and embedded devices. One of the ways to solve this problem is to put data processing near memory. Several methods such as processing in memory and processing in sensors have improved energy efficiency to some degree, but are not enough. We still need to design smarter dataflow architectures to improve data reuse and fully exploit parallelism to meet the increasing demand for more complex deep learning models on mobile devices without consuming much energy.

Innovative network architecture. Liu et al. [110] demonstrated that effective network architecture was critical to network compression and acceleration. There is a large design space to explore better but low-cost network architecture. New network architectures such as DenseNet [74] and CliqueNet [191] may be developed to achieve better performance with less complicated computations. With the rapid advancement in the area, more and more innovative deep learning models are expected to be developed in the near future. Moreover, the automatic search of network architectures to compress and accelerate the deep learning models may be more promising than the manual design of network architectures [171].

Optimization techniques combination. Existing works tackle the problem of adapting mobile learning for mobile devices from different aspects, e.g., algorithm, hardware, or software. By jointly leveraging various techniques, the performance of mobile deep learning may be further improved. There are a few pioneering works that try to unify different solutions. Across-stack optimization was utilized to reach a better performance [175], but manual tuning on across-stack optimization may be prohibitive due to a large number of diverse compression and acceleration techniques, software frameworks and hardware back-ends, and a lack of full overview across different solutions. Hence, automatic across-stack optimizations are desirable for developers to automatically pinpoint the optimal solution combination. For instance, XLA [52] and TVM [19] compilers were devised to bridge the gap between high-level frameworks and low-level hardware.

9 CONCLUSION

In this article, we provided a comprehensive survey of the state-of-the-art research on deep learning on mobile and embedded devices. We introduced relevant background information and discussed the challenges of implementing deep learning models on mobile and embedded devices. We presented a comprehensive survey of existing research that addressed the problem related to mobile deep learning from the aspects of compression & acceleration techniques, hardwares, and software frameworks. Current applications of deep learning on mobile devices were also described. From our investigation, we highlighted future directions that are critical to the success of mobile deep learning. With the rapid progress in deep learning and mobile/embedded devices, we believe that mobile deep learning will bring profound transformations in the near future.

REFERENCES

[1] Hisilicon. 2018. Hikey 970. Retrieved from https://www.96boards.org/product/hikey970/.
[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the ACM Conference on Computer and Communications Security*. 308–318.
[3] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, and François Berry. 2018. Accelerating CNN inference on FPGAs: A survey. *arXiv preprint arXiv:1806.01683* (2018).
[4] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 1–12.
[5] Moustafa Alzantot, Yingnan Wang, Zhengshuang Ren, and Mani B. Srivastava. 2017. Rstensorflow: GPU enabled tensorflow for deep learning on commodity Android devices. In *Proceedings of the ACM International Workshop on Deep Learning for Mobile Systems and Applications*. 7–12.
[6] Olov Andersson, Mariusz Wzorek, and Patrick Doherty. 2017. Deep learning quadcopter control via risk-aware active learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 3812–3818.
[7] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM J. Emerg. Technol. Comput. Syst.* 13, 3 (2017), 32.
[8] Arm. 2017. Arm compute library. Retrieved from https://www.arm.com/why-arm/technologies/compute-library.
[9] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 2654–2662.
[10] David F. Bacon, Rodric Rabbah, and Sunil Shukla. 2013. FPGA programming for the masses. *Commun. ACM* 56, 4 (2013), 56–63.

[11] Baidu. 2019. EdgeBoard. Retrieved from https://ai.baidu.com/tech/hardware/deepkit.
[12] Sourav Bhattacharya and Nicholas D. Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the ACM Conference on Embedded Network Sensor Systems*. 176–189.
[13] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the ACM Conference on Computer and Communications Security*. 1175–1191.
[14] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*. 535–541.
[15] Qingqing Cao, Niranjan Balasubramanian, and Aruna Balasubramanian. 2017. MobiRNN: Efficient recurrent neural network execution on mobile GPU. In *Proceedings of the ACM International Workshop on Deep Learning for Mobile Systems and Applications*. 1–6.
[16] Guobin Chen, Wongun Choi, Xiang Yu, Tony X. Han, and Manmohan Chandraker. 2017. Learning efficient object detection models with knowledge distillation. In *Proceedings of the Conference on Advances in Neural Information Processing Systems 30*. 742–751.
[17] Huaijin G. Chen, Suren Jayasuriya, Jiyue Yang, Judy Stephen, Sriram Sivaramakrishnan, Ashok Veeraraghavan, and Alyosha Molnar. 2016. ASP vision: Optically computing the first layer of convolutional neural networks using angle sensitive pixels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 903–912.
[18] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Not.* 49, 4 (2014), 269–284.
[19] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Q. Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the Symposium on Operating Systems Design and Implementation*. 578–594.
[20] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *Proceedings of the International Conference on Machine Learning*. 2285–2294.
[21] Xuhao Chen. 2018. Escort: Efficient sparse convolutional neural networks on GPUs. *arXiv preprint arXiv:1802.10280* (2018).
[22] Yanjiao Chen, Xueluan Gong, Qian Wang, Xing Di, and Huayang Huang. Preprint. Backdoor attacks and defenses for deep neural networks in outsourced cloud environments. *IEEE Network*. DOI: 10.1109/MNET.011.1900577
[23] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, Vol. 44. 367–379.
[24] Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu. 2018. Recent advances in efficient computation of deep convolutional neural networks. *Front. Inf. Technol. Electron. Eng.* 19, 1 (2018), 64–77.
[25] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Sig. Proc. Mag.* 35, 1 (2018), 126–136.
[26] Zhiyong Cheng, Daniel Soudry, Zexi Mao, and Zhen-zhong Lan. 2015. Training binary multilayer neural networks for image classification using expectation backpropagation. *arXiv preprint arXiv:1503.03562* (2015).
[27] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. PACT: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).
[28] François Chollet. 2016. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357* (2016).
[29] Jason Cong and Bingjun Xiao. 2014. Minimizing computation in convolutional neural networks. In *Proceedings of the International Conference on Artificial Neural Networks*. Springer, 281–290.
[30] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024* (2014).
[31] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 3123–3131.
[32] Elliot J. Crowley, Gavin Gray, and Amos J. Storkey. 2018. Moonshine: Distilling with cheap convolutions. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 2893–2903.
[33] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. 2011. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Trans. Audio Speech Lang. Proc.* 20, 1 (2011), 30–42.
[34] Li Deng. 2014. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Trans. Sig. Inf. Proc.* 3 (2014).

[35] 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 the Conference on Advances in Neural Information Processing Systems*. 1269–1277.

[36] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 4171–4186.

[37] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. 2018. Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 6797–6804.

[38] Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. 2019. HAWQ: Hessian aware quantization of neural networks with mixed-precision. *arXiv preprint arXiv:1905.03696* (2019).

[39] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, Vol. 43. 92–104.

[40] Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Arnon Amir, Brian Taba, Myron D. Flickner, and Dharmendra S. Modha. 2016. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Nat. Acad. Sci. Unit. States Amer.* 113, 41, 11441–11446.

[41] Facebook. 2017. Integrating Caffe2 on iOS/Android. Retrieved from https://caffe2.ai/docs/mobile-integration.html.

[42] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the ACM Conference on Computer and Communications Security*. 1322–1333.

[43] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. ChannelNets: Compact and efficient convolutional neural networks via channel-wise convolutions. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 5203–5211.

[44] 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. *ACM Interact. Mob. Wear. Ubiq. Technol.* 1, 3 (2017), 50:1–50:19.

[45] Robin C. Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[46] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the International Conference on Machine Learning*. 201–210.

[47] Koen Goetschalckx, Bert Moons, Patrick Wambacq, and Marian Verhelst. 2018. Efficiently combining SVD, pruning, clustering, and retraining for enhanced neural network compression. In *Proceedings of the International Workshop on Embedded and Mobile Deep Learning*. 1–6.

[48] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. 2014. A 240 G-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 682–687.

[49] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[50] Google. 2019. Coral. Retrieved from https://aiyprojects.withgoogle.com/edge-tpu/.

[51] Google. 2017. TensorFlow lite. Retrieved from https://www.tensorflow.org/lite.

[52] Google. 2017. XLA: Optimizing Complier for machine learning. Retrieved from https://www.tensorflow.org/xla.

[53] Fu-Ming Guo, Sijia Liu, Finlay S. Mungall, Xue Lin, and Yanzhi Wang. 2019. Reweighted proximal pruning for large-scale language representation. *arXiv preprint arXiv:1909.12486* (2019).

[54] Jia Guo and Miodrag Potkonjak. 2017. Pruning filters and classes: Towards on-device customization of convolutional neural networks. In *Proceedings of the ACM International Workshop on Deep Learning for Mobile Systems and Applications*. 13–17.

[55] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2017. A survey of FPGA based neural network accelerator. *arXiv preprint arXiv:1712.08934* (2017).

[56] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient DNNs. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 1379–1387.

[57] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the International Conference on Machine Learning*. 1737–1746.

[58] P. Gysel, M. Motamedi, and S. Ghiasi. 2016. Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1604.03168* (2016).

[59] Ramyad Hadidi, Jiashen Cao, Yilun Xie, Bahar Asgari, Tushar Krishna, and Hyesoon Kim. 2019. Characterizing the deployment of deep neural networks on commercial edge devices. In *Proceedings of the IEEE International Symposium on Workload Characterization*. 35–48.

[60] 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. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*. 243–254.

[61] 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).

[62] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 1135–1143.

[63] William Grant Hatcher and Wei Yu. 2018. A survey of deep learning: Platforms, applications, and emerging research trends. *IEEE Access* 6 (2018), 24411–24432.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[65] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 2234–2240.

[66] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 784–800.

[67] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 1398–1406.

[68] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. 2019. Knowledge distillation with adversarial samples supporting decision boundary. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3771–3778.

[69] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[70] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: Information leakage from collaborative deep learning. In *Proceedings of the ACM Conference on Computer and Communications Security*. 603–618.

[71] 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).

[72] Hengyuan Hu, Rui Peng, Yu Wing Tai, and Chi Keung Tang. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250* (2016).

[73] Qinghao Hu, Peisong Wang, and Jian Cheng. 2018. From hashing to CNNs: Training binary weight networks via hashing. *arXiv preprint arXiv:1802.02733* (2018).

[74] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2261–2269.

[75] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 4107–4115.

[76] 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. *J. Mach. Learn. Res.* 18, 1 (2017), 6869–6898.

[77] Loc Nguyen Huynh, Rajesh Krishna Balan, and Youngki Lee. 2016. DeepSense: A GPU-based deep convolutional neural network framework on commodity mobile devices. In *Proceedings of the ACM Workshop on Wearable Systems and Applications*. 25–30.

[78] Loc N. Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. DeepMon: Mobile GPU-based deep learning framework for continuous vision applications. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services*. 82–95.

[79] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[80] Intel. 2017. Movidius Myriad X VPU. Retrieved from https://www.movidius.com/myriadx.

[81] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *Proceedings of the Symposium on VLSI Technology*. 87–88.

[82] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling BERT for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).

[83] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture*. 1–12.

[84] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (high bandwidth memory) DRAM technology and architecture. In *Proceedings of the IEEE International Memory Workshop*. 1–4.

[85] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A low latency framework for secure neural network inference. In *Proceedings of the USENIX Security Symposium*. 1651–1669.

[86] Doris Keitel-Schulz and Norbert Wehn. 2001. Embedded DRAM development: Technology, physical design, and application issues. *IEEE Des. Test Comput.* 18, 3 (2001), 7–15.

[87] Beomjun Kim, Yongsu Jeon, Heejin Park, Dongheon Han, and Yunju Baek. 2017. Design and implementation of the vehicular camera system using deep neural network compression. In *Proceedings of the ACM International Workshop on Deep Learning for Mobile Systems and Applications*. 25–30.

[88] Kye-Hyeon Kim, Sanghoon Hong, Byungseok Roh, Yeongjae Cheon, and Minje Park. 2016. PVANET: Deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv:1608.08021* (2016).

[89] Minje Kim and Paris Smaragdis. 2016. Bitwise neural networks. *arXiv preprint arXiv:1601.06071* (2016).

[90] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Proceedings of the International Conference on Learning Representations*.

[91] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the International Conference on Neural Information Processing Systems*. 1097–1105.

[92] Griffin Lacey, Graham W. Taylor, and Shawki Areibi. 2016. Deep learning on FPGAs: Past, present, and future. *arXiv preprint arXiv:1602.04283* (2016).

[93] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[94] 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 the ACM/IEEE International Conference on Information Processing in Sensor Networks*. 1–12.

[95] Seyyed Salar Latifi Oskouei, Hossein Golestani, Matin Hashemi, and Soheil Ghiasi. 2016. CNNdroid: GPU-accelerated execution of trained deep convolutional neural networks on Android. In *Proceedings of the ACM International Conference on Multimedia*. 1201–1205.

[96] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4013–4021.

[97] Yann Le Cun, John S. Denker, and Sara A. Solla. 1990. Optimal brain damage. *Adv. Neural Inf. Proc. Syst.* 2, 279 (1990), 598–605.

[98] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. 2015. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *Proceedings of the International Conference on Learning Representations*.

[99] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. 2018. Extremely low bit neural network: Squeeze the last bit out with ADMM. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 3466–3473.

[100] Fengfu Li and Bin Liu. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).

[101] Guiying Li, Chao Qian, Chunhui Jiang, Xiaofen Lu, and Ke Tang. 2018. Optimization based layer-wise magnitude-based pruning for DNN compression. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 2383–2389.

[102] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient ConvNets. In *Proceedings of the International Conference on Learning Representations*.

[103] Xiang Li, Tao Qin, Jian Yang, Xiaolin Hu, and Tieyan Liu. 2016. LightRNN: Memory and computation-efficient recurrent neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 4385–4393.

[104] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. 2016. RedEye: Analog ConvNet image sensor architecture for continuous mobile vision. In *ACM SIGARCH Computer Architecture News*, Vol. 44. 255–266.

[105] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. *arXiv preprint arXiv:1312.4400* (2013).

[106] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. 2018. Accelerating convolutional networks via global & dynamic filter pruning. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 2425–2432.

[107] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the ACM Conference on Computer and Communications Security*. 619–631.

[108] Xingyu Liu, Jeff Pool, Song Han, and William J. Dally. 2018. Efficient sparse-Winograd convolutional neural networks. In *Proceedings of the International Conference on Learning Representations*.

[109] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*. 2755–2763.

[110] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2019. Rethinking the value of network pruning. In *Proceedings of the International Conference on Learning Representations*.

[111] Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong. 2018. Frequency-domain dynamic pruning for convolutional neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 1051–1061.

[112] Huynh Nguyen Loc, Youngki Lee, and Rajesh Krishna Balan. 2018. D-Pruner: Filter-based pruning method for deep convolutional neural network. In *Proceedings of the International Workshop on Embedded and Mobile Deep Learning*. 7–12.

[113] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3431–3440.

[114] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. ThiNet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*. 5068–5076.

[115] Jian-Hao Luo and Jianxin Wu. 2018. AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941* (2018).

[116] Jiancheng Lyu, Shuai Zhang, Yingyong Qi, and Jack Xin. 2019. AutoShuffleNet: Learning permutation matrices via an exact Lipschitz continuous penalty in deep convolutional neural networks. *CoRR* abs/1901.08624 (2019).

[117] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *Proceedings of the European Conference on Computer Vision*. 122–138.

[118] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. 2017. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922* (2017).

[119] Manu Mathew, Kumar Desappan, Pramod Kumar Swami, and Soyeb Nagori. 2017. Sparse, quantized, full frame CNN for low power embedded devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 11–19.

[120] Ian McGraw, Rohit Prabhavalkar, Raziel Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Haşim Sak, Alexander Gruenstein, Françoise Beaufays, et al. 2016. Personalized speech recognition on mobile devices. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. 5955–5959.

[121] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the Conference on Artificial Intelligence and Statistics*. 1273–1282.

[122] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning differentially private recurrent language models. In *Proceedings of the International Conference on Learning Representations*.

[123] Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. 2017. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462* (2017).

[124] Austin Meyers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin P. Murphy. 2015. Im2Calories: Towards an automated mobile vision food diary. In *Proceedings of the IEEE International Conference on Computer Vision*. 1233–1241.

[125] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of the Conference of the International Speech Communication Association*. 1045–1048.

[126] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025* (2016).

[127] Mehdi Mohammadi, Ala I. Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. 2018. Deep learning for IoT big data and streaming analytics: A survey. *IEEE Commun. Surv. Tutor.* 20, 4 (2018), 2923–2960.

[128] Payman Mohassel and Peter Rindal. 2018. ABY 3: A mixed protocol framework for machine learning. In *Proceedings of the ACM Conference on Computer and Communications Security*. 35–52.

[129] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *Proceedings of the IEEE Symposium on Security and Privacy*. 19–38.

[130] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440* (2016).

[131] David Moloney, Brendan Barry, Richard Richmond, Fergal Connor, Cormac Brick, and David Donohoe. 2014. Myriad 2: Eye of the computational vision storm. In *Proceedings of the IEEE Hot Chips 26 Symposium*. 1–18.

[132] Matthew W. Moskewicz, Forrest N. Iandola, and Kurt Keutzer. 2016. Boda-RTC: Productive generation of portable, efficient code for convolutional neural networks on mobile computing platforms. In *Proceedings of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. 1–10.

[133] Mohammad Motamedi, Daniel Fong, and Soheil Ghiasi. 2016. Fast and energy-efficient CNN inference on IoT devices. *arXiv preprint arXiv:1611.07151* (2016).

[134] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. 2016. Design space exploration of FPGA-based deep convolutional neural networks. In *Proceedings of the IEEE Asia and South Pacific Design Automation Conference*. 575–580.

[135] Mohammad Motamedi, Philipp Gysel, and Soheil Ghiasi. 2017. PLACID: A platform for FPGA-based accelerator creation for DCNNs. *ACM Trans. Multimedia Comput. Commun. Applic.* 13, 4 (2017).

[136] Cedric Nugteren. 2018. CLBlast: A tuned OpenCL BLAS library. In *Proceedings of the International Workshop on OpenCL*. 5:1–5:10.

[137] Nvidia. 2019. Jetson Nano. Retrieved from https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/.

[138] Kaoru Ota, Minh Son Dao, Vasileios Mezaris, and Francesco G. B. De Natale. 2017. Deep learning for mobile multimedia: A survey. *ACM Trans. Multimedia Comput. Commun. Applic.* 13, 3s (2017), 34.

[139] 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. In *Proceedings of the ACM International Symposium on Computer Architecture*. 27–40.

[140] Jongsoo Park, Sheng R. Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. 2017. Faster CNNs with direct sparse convolutions and guided pruning. In *Proceedings of the International Conference on Learning Representations*.

[141] NhatHai Phan, Yue Wang, Xintao Wu, and Dejing Dou. 2016. Differential privacy preservation for deep auto-encoders: An application of human behavior prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 1309–1316.

[142] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. In *Proceedings of the International Conference on Learning Representations*.

[143] A. Polyak and L. Wolf. 2015. Channel-level acceleration of deep face representations. *IEEE Access* 3 (2015), 2163–2175.

[144] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2018. A survey on deep learning: Algorithms, techniques, and applications. *Comput. Surv.* 51, 5 (2018), 92.

[145] Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, and Lan McGraw. 2016. On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. 5970–5974.

[146] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 26–35.

[147] Valentin Radu, Kuba Kaszyk, Yuan Wen, Jack Turner, José Cano, Elliot J. Crowley, Björn Franke, Amos J. Storkey, and Michael O'Boyle. 2019. Performance aware convolutional neural network channel pruning for embedded GPUs. In *Proceedings of the IEEE International Symposium on Workload Characterization*. 24–34.

[148] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*. Springer, 525–542.

[149] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proceedings of the Conference on International Conference on Neural Information Processing Systems*. 91–99.

[150] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based oblivious deep neural network inference. In *Proceedings of the USENIX Security Symposium*. 1501–1518.

[151] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. FitNets: Hints for thin deep nets. In *Proceedings of the International Conference on Learning Representations*.

[152] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536.

[153] Tara N. Sainath, Brian Kingsbury, Hagen Soltau, and Bhuvana Ramabhadran. 2013. Optimization techniques to improve training speed of deep neural networks for large speech tasks. *IEEE Trans. Audio, Speech Lang. Proc.* 21, 11 (2013), 2267–2276.

[154] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection, and segmentation. *arXiv preprint arXiv:1801.04381* (2018).

[155] Sayantan Sarkar, Vishal M. Patel, and Rama Chellappa. 2016. Deep feature-based face detection on mobile devices. In *Proceedings of the IEEE International Conference on Identity, Security and Behavior Analysis*. 1–8.

[156] Syed Shakib Sarwar, Priyadarshini Panda, and Kaushik Roy. 2017. Gabor filter assisted energy efficient fast learning convolutional neural networks. In *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*. 1–6.

[157] Bharat Bhusan Sau and Vineeth N. Balasubramanian. 2016. Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650* (2016).

[158] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. Q-BERT: Hessian based ultra low precision quantization of BERT. *arXiv preprint arXiv:1909.05840* (2019).

[159] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *Proceedings of the IEEE Symposium on Security and Privacy*. 3–18.

[160] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*.

[161] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P. Namboodiri. 2019. HetConv: Heterogeneous kernel-based convolutions for deep CNNs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4835–4844.

[162] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Machine learning models that remember too much. In *Proceedings of the ACM Conference on Computer and Communications Security*. 587–601.

[163] Guillaume Soulié, Vincent Gripon, and Maëlys Robert. 2016. Compression of deep neural networks on the fly. In *Proceedings of the International Conference on Artificial Neural Networks*. Springer, 153–160.

[164] Suraj Srinivas and R. Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. In *Proceedings of the British Machine Vision Conference*. 31.1–31.12.

[165] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 16–25.

[166] Ke Sun, Mingjie Li, Dong Liu, and Jingdong Wang. 2018. IGCV3: Interleaved low-rank group convolutions for efficient deep neural networks. In *Proceedings of the British Machine Vision Conference*. 101.

[167] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. *arXiv preprint arXiv:1908.09355* (2019).

[168] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[169] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[170] Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. 2016. Convolutional neural networks with low-rank regularization. In *Proceedings of the International Conference on Learning Representations*.

[171] 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 the IEEE Conference on Computer Vision and Pattern Recognition*. 2820–2828.

[172] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from BERT into simple neural networks. *arXiv preprint arXiv:1903.12136* (2019).

[173] Ryosuke Tanno, Koichi Okamoto, and Keiji Yanai. 2016. DeepFoodCam: A DCNN-based real-time mobile food recognition system. In *Proceedings of the International Workshop on Multimedia Assisted Dietary Management*. 89–89.

[174] Frederick Tung and Greg Mori. 2018. CLIP-Q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7873–7882.

[175] Jack Turner, José Cano, Valentin Radu, Elliot J. Crowley, Michael O'Boyle, and Amos J. Storkey. 2018. Characterising across-stack optimisations for deep convolutional neural networks. In *Proceedings of the IEEE International Symposium on Workload Characterization*. 101–110.

[176] Amund Tveit, Torbjørn Morland, and Thomas Brox Røst. 2016. DeepLearningKit—A GPU optimized deep learning framework for Apple's iOS, OS X, and tvOS developed in Metal and Swift. *arXiv preprint arXiv:1605.04614* (2016).

[177] Ganesh Venkatesh, Eriko Nurvitadhi, and Debbie Marr. 2017. Accelerating deep convolutional networks using low-precision and sparsity. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. 2861–2865.

[178] Huan Wang, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. 2019. Structured pruning for efficient ConvNets via incremental regularization. In *Proceedings of the International Joint Conference on Neural Networks*. 1–8.

[179] Ji Wang, Weidong Bao, Lichao Sun, Xiaomin Zhu, Bokai Cao, and S. Yu Philip. 2019. Private model compression via knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 1190–1197.

[180] Peisong Wang and Jian Cheng. 2017. Fixed-point factorized networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3966–3974.

[181] Peiqi Wang, Xinfeng Xie, Lei Deng, Guoqi Li, Dongsheng Wang, and Yuan Xie. 2018. HitNet: Hybrid ternary recurrent neural network. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 602–612.

[182] Zhibo Wang, Mengkai Song, Siyan Zheng, Zhifei Zhang, Yang Song, and Qian Wang. 2019. Invisible adversarial attack against deep neural networks: An adaptive penalization approach. *IEEE Trans. Depend. Sec. Comput.* 99 (2019), 1–1, 10.1109/TDSC.2019.2929047.

[183] Tang Wei, Hua Gang, and Wang Liang. 2017. How to train a compact binary neural network with high accuracy? In *Proceedings of the AAAI Conference on Artificial Intelligence*. 2625–2631.

[184] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 2074–2082.

[185] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4820–4828.

[186] Z. Xiangyu, Z. Xinyu, L. Mengxiao, and S. Jian. 2017. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*. 6848–6856.

[187] Guotian Xie, Jingdong Wang, Ting Zhang, Jianhuang Lai, Richang Hong, and Guo-Jun Qi. 2018. Interleaved structured sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8847–8856.

[188] XILINX. 2020. FPGA vs. ASIC. Retrieved from https://www.xilinx.com/products/silicon-devices/fpga.html.

[189] Haichuan Yang, Yuhao Zhu, and Ji Liu. 2018. End-to-end learning of energy-constrained deep neural networks. *CoRR* abs/1806.04321 (2018).

[190] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6071–6079.

[191] Yibo Yang, Zhisheng Zhong, Tiancheng Shen, and Zhouchen Lin. 2018. Convolutional neural networks with alternately updated clique. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2413–2422.

[192] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. 2015. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*. 1476–1483.

[193] Jieping Ye. 2005. Generalized low rank approximations of matrices. *Mach. Learn.* 61, 1–3 (2005), 167–191.

[194] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. 2017. A gift from knowledge distillation: Fast optimization, network minimization, and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7130–7138.

[195] Sergey Zagoruyko and Nikos Komodakis. 2017. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *Proceedings of the International Conference on Learning Representations*.

[196] Xiao Zeng, Kai Cao, and Mi Zhang. 2017. MobileDeepPill: A small-footprint mobile deep learning system for recognizing unconstrained pill images. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services*. 56–67.

[197] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 161–170.

[198] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep learning in mobile and wireless networking: A survey. *IEEE Commun. Surv. Tutor.* 21, 3 (2019), 2224–2287.

[199] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision*. 373–390.

[200] Jintao Zhang, Zhuo Wang, and Naveen Verma. 2016. A machine-learning classifier implemented in a standard 6T SRAM array. In *Proceedings of the IEEE Symposium on VLSI Circuits*. 1–2.

[201] Qingchen Zhang, Laurence T. Yang, Xingang Liu, Zhikui Chen, and Peng Li. 2017. A Tucker deep computation model for mobile multimedia feature learning. *ACM Trans. Multimedia Comput. Commun. Applic.* 13, 3s (2017), 39.

[202] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 1–12.

[203] Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. 2017. Interleaved group convolutions for deep neural networks. *CoRR* abs/1707.02725 (2017).

[204] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. 2015. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 38, 10 (2015), 1943–1955.

[205] Lingchen Zhao, Shengshan Hu, Qian Wang, Jianlin Jiang, Chao Shen, Xiangyang Luo, and Pengfei Hu. Preprint. Shielding collaborative learning: Mitigating poisoning attacks through client-side detection. *IEEE Trans. Depend. Sec. Comput.* DOI: 10.1109/TDSC.2020.2986205

[206] Lingchen Zhao, Qian Wang, Qin Zou, Yan Zhang, and Yanjiao Chen. 2019. Privacy-preserving collaborative deep learning with unreliable participants. *IEEE Trans. Inf. Forens. Sec.* 15 (2019), 1486–1500, DOI: 10.1109/TIFS.2019.2939713

[207] Sanqiang Zhao, Raghav Gupta, Yang Song, and Denny Zhou. 2019. Extreme language model compression with optimal subwords and shared projections. *arXiv preprint arXiv:1909.11687* (2019).

[208] Zejia Zhengj and Juyang Weng. 2016. Mobile device based outdoor navigation with on-line learning neural network: A comparison with convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 11–18.

[209] Jing Zhong, Guiguang Ding, Yuchen Guo, Jungong Han, and Bin Wang. 2018. Where to prune: Using LSTM to guide end-to-end pruning. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 3205–3211.

[210] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *Proceedings of the International Conference on Learning Representations*.

[211] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[212] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2017. Trained ternary quantization. In *Proceedings of the International Conference on Learning Representations*.

[213] Michael Zhu and Suyog Gupta. 2018. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *Proceedings of the International Conference on Learning Representations Workshop*.

[214] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jin-Hui Zhu. 2018. Discrimination-aware channel pruning for deep neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*. 883–894.

Received April 2019; revised March 2020; accepted May 2020