Neurocomputing 614 (2025) 128791

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Survey Paper

A review of AI edge devices and lightweight CNN and LLM deployment

Kailai Sun [a,b,*,1], Xinwei Wang [b,1], Xi Miao [c], Qianchuan Zhao [b]

[a] College of Design and Engineering, National University of Singapore, 4 Architecture Drive, 117566, Singapore
[b] Center for Intelligent and Networked Systems, Department of Automation, BNRist, Tsinghua University, Beijing, 100084, China
[c] Penn State University, Department of Mathematics, 201 Old Main Street, University Park, PA, 16802, United States

ARTICLE INFO

Communicated by N. Passalis

Keywords:
AI edge devices
Practical deployment
Lightweight deep neural networks
Neural network compression

ABSTRACT

Artificial Intelligence of Things (AIoT) which integrates artificial intelligence (AI) and the Internet of Things (IoT), has attracted increasing attention recently. With the remarkable development of AI, convolutional neural networks (CNN) have achieved great success from research to deployment in many applications. However, deploying complex and state-of-the-art (SOTA) AI models on edge applications is increasingly a big challenge. This paper investigates literature that deploys lightweight CNNs on AI edge devices in practice. We provide a comprehensive analysis of them and many practical suggestions for researchers: how to obtain/design lightweight CNNs, select suitable AI edge devices, and compress and deploy them in practice. Finally, future trends and opportunities are presented, including the deployment of large language models, trustworthy AI and robust deployment.

1. Introduction

With the rapid development of the Internet of Things (IoT), there will be 41.6 billion connected IoT devices, generating nearly 79.4 zettabytes (ZB) of data in 2025, estimated by International Data Corporation (IDC) [1]. Such large volumes of data need to be processed through smart methods. Artificial intelligence (AI) technologies, especially deep learning (DL), have achieved remarkable processes in data analysis, prediction, and decision-making [2]. Artificial Intelligence of Things (AIoT), which integrates AI and the IoT [3], has become the research spotlight.

AI technologies have been widely applied in many fields with excellent performance in recent years, including natural language processing (NLP) [4], image classification [5], face (object) detection [6], video-based object segmentation [7], applied mathematics [8], building control [9], etc. But deep neural networks (DNNs) also face great challenges in practical applications. In general, a deeper and more complex network brings a higher upper bound of performance, but it will also bring a practical problem: high demands on the devices' computing power, memory size, and power consumption. The early LeNet [10] with only 5 layers, can be directly deployed on central processing units (CPUs). But most current mainstream networks (e.g., Resnet [11] and Transformer [12]) can only achieve acceptable inference speeds when deployed on high-performance graphics processing units (GPUs) (e.g., Nvidia RTX/GTX series), which are expensive and have high power consumption.

Due to the complex AI tasks and high computing demands, cloud-based AIoT generally applies high-performance cloud servers to analyze collected data from IoT devices using AI technologies [13], for achieving a better service. However, the long distance between the cloud center and IoT devices brings many problems: the serious pressure on network bandwidth; the inherent latency of network communication; the exposure of private information [14]. To improve the Quality of Experience (QoE) and address the challenges in AIoT applications, intelligent edge computing is proposed by deploying AI accelerators in the edge [15].

When applying AI models in practical scenarios, deep neural networks are usually deployed on cheap, low-power, and small-edge devices. It brings a new problem: if a mainstream convolutional neural network (CNN), is directly deployed on these devices, the inference speed will be unacceptable, and even the memory will not be enough. Therefore, how to deploy a lightweight CNN on AI edge devices while ensuring its performance has become a very urgent and meaningful problem.

Based on these observations, this article investigates the convergence of lightweight convolutional neural networks, neural network compression, and AI edge devices, and is interested in techniques enabling efficient and effective deployment in practice. Although some recent reviews of IoT exist [3,15,16], their coverage of lightweight CNNs compression is limited. Compared with recent surveys on CNNs

* Corresponding author.
E-mail addresses: skl23@nus.edu.sg (K. Sun), wxw21@mails.tsinghua.edu.cn (X. Wang).
[1] These authors contributed equally to this work.

https://doi.org/10.1016/j.neucom.2024.128791
Received 6 August 2024; Received in revised form 27 September 2024; Accepted 19 October 2024
Available online 28 October 2024

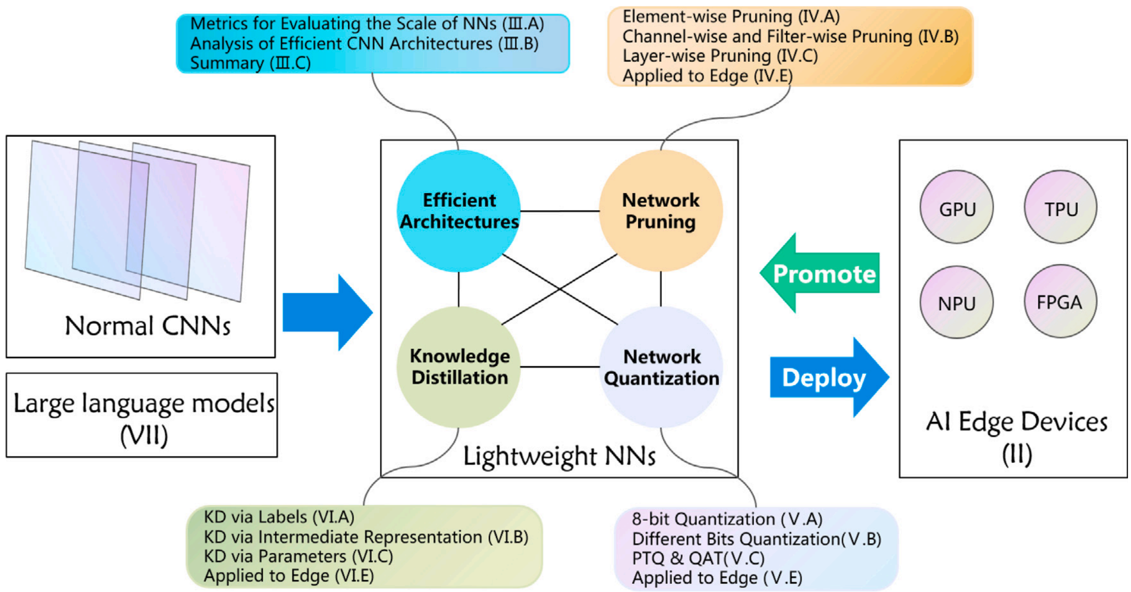

Fig. 1. Overview of lightweight deployment for CNNs. KD: Knowledge Distillation. PTQ: post-training quantization. QAT: quantization-aware training.

compression [17–19] and hardware [20], we present a more practical summary of most existing methodologies, showing more detailed technical solutions of lightweight deployment, and talk about both algorithm and hardware synergistically. Unlike these survey/review papers, our paper focuses on the practical value, aiming to present a more practical summary by reviewing a lot of research work which deploys AI devices with neural network compression in practical scenes. Unlike these survey/review papers, this article investigates related studies that have successfully deployed lightweight CNNs on AI edge devices in practical perception. This work aims to conduct a review to provide some suggestions to researchers: how to obtain/design lightweight CNNs, how to select suitable AI edge devices, and how to compress and deploy them in practice. In summary, the contributions of this work can be summarized as follows.

- We focus on lightweight CNNs, neural network compression, and their deployment on AI edge devices for practical applications, and review many recent, especially five-year studies.
- Many typical AIoT edge devices with AI accelerators are compared and analyzed in Tables 1, 2, 3, 4 and 5.
- By taking a more global approach, we present four comprehensive metrics to evaluate efficient architectures.
- To bridge the gap between state-of-the-art (SOTA) AI models and practical deployment, we conduct an analysis on studies that apply individual neural network compression methods in practical applications in Tables 6, 7, 8, and apply fused neural network compression methods in Table 9.
- The deployment of large language models is briefly reviewed and analyzed.
- Detailed discussion and suggestions are provided for researchers. Some future trends are presented, including trustworthy AI and robust deployment.

In Fig. 1, the overview of lightweight deployment is presented. First, many normal CNNs are necessary to be compressed when they are applied in edge tasks. Second, four lightweight methods are presented. They have different advantages and limitations. Efficient architectures can provide the initial architectural design choices, considering accuracy, speed and memory metrics. Network pruning can reduce redundant connections while maintaining effective connections in network architectures. Network quantization quantizes the float tensor to a lower number of digits. Knowledge distillation applies the 'teacher–student' way to distil the knowledge learned by a large network to a small network. Third, when deploying lightweight CNNs in practice, selecting suitable AI edge devices is very important. Many AI accelerators (e.g., GPU, TPU, NPU) need to be considered and optimized to meet practical requirements. Fourth, on the other hand, AI edge devices will promote further network adjustment in turn. Last, This loop ends until the user's needs are satisfied.

The remaining parts of the paper are organized as follows. Section 2 mainly analyzes and compares many AI edge devices in different aspects. Section 3 presents four metrics to evaluate lightweight CNNs, and analyzes the structure and design of lightweight CNNs. Sections 4, 5, 6 introduce the network pruning, network quantization, and knowledge distillation technologies in detail, respectively. Sections 7, 8 discusses and compares the advantages and limitations of different methods and presents some future trends. Finally, Section 9 concludes this review work.

2. AI edge devices

The concepts of 'edge' and 'cloud' come from the Internet of Things (IoT). The cloud could be understood as clusters of data centers and servers that **centrally** process and store large amounts of data. The edge refers to **local** devices that connect and control other end devices such as mobile phones, robots, smart home appliances, sensors, and other devices with a network interface controller (NIC). Then humans can control other end devices or collect data using a single device without physically contacting other people or other devices. Under this definition, the scope of edge devices is broad. Here we use the 'AI Edge Device' concept to limit the range. The 'AI Edge Devices' discussed in this paper meet the following criteria:

- GPUs or other computing units optimized for deep learning.
- Exchanging information with other devices remotely.
- Low power consumption.
- Small size.
- Cheap.
- Programmable to deal with different tasks.

The small size means that the area of the motherboard cannot be too large because the motherboard area mainly determines the size of these devices (see Fig. 2).

Under this definition, the AI edge devices discussed in this paper are limited to small computers dedicated to edge AI deployment.

Table 1
Performance parameters of edge devices.

Company	Nvidia			Intel
Device	Nano	TX2	Xavier NX	NCS2[a]
Computing power	0.472TFLOPS	1.33TFLOPS	21TOPS	1TOPS
Power Consumption	10 W	15 W	20 W	1 W
CPU	4×ARM Cortex-A57 MPCore CPU	2×NVIDIA Denver 2 64 bit CPU and 4×Arm Cortex-A57 MPCore CPU	6×NVIDIA Carmel ARM v8.2 64 bit CPU	*
GPU	Maxwell architecture, contains 128 CUDA cores	Pascal architecture,256 CUDA cores	Volta architecture,384 CUDA cores, and 64 Tensor cores	*
AI accelerator	*	*	2×NVLDA engine	Movidius VPU
RAM	4 GB 64 bit 1600 MHz LPDDR4 25.6 GB/s	8 GB 128 bit LPDDR4 1866Mhz 59.7 GB/s	8/16 GB 128 bit LPDDR4 1866Mhz 59.7 GB/s	*
Supported DL frameworks	TensorFlow, PyTorch, MxNet, Keras	TensorFlow, PyTorch, MxNet, Keras	TensorFlow, PyTorch, MxNet, Keras	Openvino[b]

[a] Neural compute stick (NCS) 2, Coral USB and MLU220-M.2 are coprocessors that only contain computing units. They need another small computer like Raspberry Pi to work.
[b] We cannot directly deploy Tensorflow, PyTorch, or other DL frameworks on NCS2. However, we can use Openvino (DL framework specially designed for Intel's device) to apply trained models from other DL frameworks for inference. Now Openvino supports Tensorflow, PyTorch, MXNet, Caffe, ONNX, and Kaldi models.

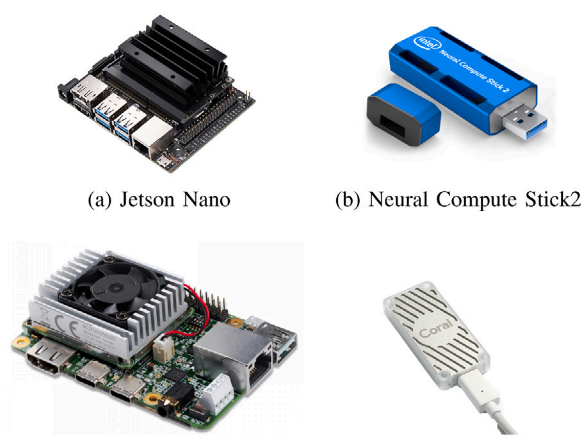
(a) Jetson Nano (b) Neural Compute Stick2
(c) Coral Dev Board (d) Coral USB Accelerator

Fig. 2. Edge devices.

Some coprocessors that only retain computing units for deep learning (e.g., tensor process units (TPU) or neural-network process units (NPU)) are also within this scope. It is worth mentioning that the advancement in chips has led to enhanced computing power in mobile phones, rendering some of them (e.g., iPhones) capable of supporting AI deployment. Consequently, they are also regarded as AI edge devices.

Common AI edge devices and their main parameters are listed in Tables 1, 2, 3, 4 and 5. We compare them in terms of company, power consumption, CPU, and supported DL frameworks, etc. Computing power is the core factor for measuring the computing speed of edge devices. It has two units: operations per second (OPS) and floating-point operations per second (FLOPS). TOPS means tera (10^{12}) operations per second. The same goes for TFLOPS. We can see that most AI edge devices use TOPS as the computing power unit. Because the operations of TOPS are Int8 operations, most electronic devices support Int8 operations. However, CNNs use floating-point calculations by default, so FLOPS can better reflect the device's computing power in the deep learning field. AI accelerators represent hardware units specifically designed for deep learning acceleration. RAM means random-access memory.

Based on the information provided in Tables 1, 2, 3, 4 and 5, we can conclude:

1. Most AI edge devices have low computing power and power consumption, which is a practical trade-off between cost and performance requirements. This suggests that AI edge devices are designed to be power-efficient and cost-effective for practical deployment.

2. Due to practical low-power requirements, many AI edge devices use Arm's low-power, high-performance CPUs. Arm's CPUs are optimized for power efficiency, making them suitable for AI edge devices.

3. AI edge devices use a variety of GPUs and AI accelerators for model inference. GPUs and AI accelerators are the core source of computing power.

4. The size of RAM available on an AI edge device is critical because it determines the size of the CNN that can run on the device. However, since the CPU and GPU typically share the RAM in most AI edge devices, more RAM is preferable for deployment. Memory bandwidth (Frequency × Bits/8) of RAM is another critical factor that determines the speed at which the computing units (GPU, NPU) can read and write data and weights from RAM. It also significantly affects the inference speed of AI edge devices.

5. The supported DL framework is an important factor for users of AI edge devices. If a user has a mature model trained under a specific DL framework, but the AI edge device he uses does not support that framework, migrating the model to another framework can be very difficult. Therefore, it is essential to choose an AI edge device that supports the DL framework required for specific requirements.

Choosing an appropriate AI edge device depends on the practical requirements. Here are some suggestions on how to choose an AI edge device based on different scenarios:

1. If the task is simple image classification and the application scenario requires low cost and power consumption, Intel's NCS2 is a good choice. Its low power consumption of 1 W and computing power of 1TOPS make it suitable for this scenario. Besides, NCS2 supports DL frameworks such as TensorFlow, PyTorch, and ONNX through its OpenVino, which is easy to deploy deep models.

2. The Coral Dev Board is a good choice if high inference speed and low power consumption are required. It has the highest ratio of computing power divided by power consumption among all the AI edge devices in the tables. Due to engineers' specific optimization, its CNN's inference speed is much higher than that of other devices with the same computing power. However, the Coral Dev Board only supports TensorFlow Lite so far, which may not be user-friendly.

3. The Jetson series is a good choice if power consumption is not a significant concern. Jetson Nano has the highest computing power among the three devices (NCS2, Jetson Nano and Coral Dev

Table 2
Performance parameters of edge devices.

Company	Google		Baidu	AMD	Huawei
Device	Coral Dev Board	Coral USB	Edgeboard FZ3	Kria K26	Atlas200 DK
Computing power	4TOPS	4TOPS	1.2TOPS	1.4TOPS	22TOPS
Power Consumption	2 W	2 W	12 W	15 W	36 W
CPU	4xCortex-A53, Cortex-M4F	*	4xCortex-A53	4xCortex-A53, Cortex-R5F	Hi3559A
GPU	Integrated GC7000 Lite Graphics	*	*	Arm Mali™-400MP2	*
AI accelerator	Google Edge TPU coprocessor	Google Edge TPU coprocessor	*	FPGA	HUAWEI Ascend 310
RAM	1 GB LPDDR4	*	2(4) GB 32(64) bit 2400 MHz LPDDR4	4 GB 64 bit DDR4	4(8)GB 128(256) bit 3200 MHz LPDDR4
Supported DL frameworks	Tensorflow lite	Tensorflow Lite	EazyDL, BML, Paddle	Vitis AI	Tensorflow, Caffe, Pytorch

Table 3
Performance parameters of edge devices.

Company	Sophon	Edgelock	Cambricon	
Device	SM5	i.MX 8 plus	MLU220-SOM	MLU220-M.2
Computing power	17.6(35.2)TOPS	2.3TOPS	16TOPS	8TOPS
Power Consumption	25 W	*	15 W	8.25 W
CPU	ARM 8-core A53	Cortex-A53 × 2(4)	ARM A55 × 4	*
GPU	*	GC7000UL, GC520L	*	*
AI accelerator	BM1684 TPU	NPU	MLUv02	MLUv02
RAM	12 GB	32 bit LPDDR4	8 GB 64 bit 3733 MHz LPDDR4	*
Supported DL frameworks	Caffe, Tensorflow, PyTorch, Paddle, Mxnet	eIQ (support DeepviewRT, Tensorflow Lite (Micro), CMSIS-NN and Glow)	Tensorflow, PyTorch, Caffe, Mxnet	Tensorflow, PyTorch, Caffe, Mxnet

Table 4
Performance parameters of edge devices.

Company	Rockchip	Qualcomm	Apple	MediaTek	Hailo
Device	RK3566	Snapdragon 8gen3	A17pro	Dimensity 9300	Hailo-8 M.2 AI Acceleration Module
Computing power	1TOPS	4.6 TFLOPS	35TOPS	*	26TOPS
Power Consumption	5 W	10 W	11 W	10 W	2.5 W
CPU	Cortex-A55 × 4	Cortex-X4 + 5 × A720 + 2 × A520	A17	4x Arm Cortex-X4 at 3.25 GHz + 4x Arm Cortex-A720	*
GPU	Mali-G52	Adreno	A17 Pro	Arm Immortalis-G720 MC12	*
AI accelerator	NPU	Hexagon	Apple neural Engine	MediaTek APU 790	Hailo-8
RAM	4 GB	*	6 GB GPU memory	*	*
Supported DL frameworks	Caffe, Tensorflow, PyTorch, Mxnet	Pytorch	MLX	NeuroPilot	TensorFLow, ONNX, Keras, Pytorch

Board) mentioned and supports directly deploying TensorFlow, PyTorch, MxNet, and Keras. It also achieves 16-bit floating-point computing acceleration at the hardware level. The cost of these advantages is that Jetson Nano's power consumption is much higher than that of NCS2 and Coral Dev Board.

Therefore, when choosing an AI edge device, it is important to consider the application's specific requirements, such as power consumption, computing power, inference speed, and DL framework support, to find the most appropriate device.

As we can see from above, many factors affect the running speed of deep neural networks on edge devices, but the core factor would be the chip. The chip's manufacturing technology, architecture design, etc., will significantly affect the device's performance. The impact of the manufacturing technology on performance is direct: the operation of the chip depends on the 1/0 voltage changes of the transistors on the chip. So more transistors will lead to stronger computing power [21]. A more advanced manufacturing technology makes transistors smaller. Smaller transistors can increase transistors amount per cm^2, then achieve stronger computing power. As for the complex architecture of the chip, we take the Nvidia's Jetson AGX ORIN as an example in Fig. 3: it has three core parts: 3 sets of 4-core ARM Cortex CPUs, Ampere architecture 2048-cores GPU with 64 Tensor acceleration cores, and 2 NVDLA2 accelerators dedicated to deep learning acceleration. The CPU is responsible for complex tasks and controlling other components (e.g., the GPU). The GPU is designed for graphics computing tasks, using the advantages of multi-core and multi-threading to complete massive but simple graphics computing tasks. Nvidia's compute unified device architecture (CUDA) cores enable the GPU to deal with more complex tasks than simple graphics calculations,

Table 5
Performance parameters of edge devices.

Company	AMD			Lattice	Xilinx
Device	Corazon-AI	Spartan-7-SP701	Versa-2VE3304	Avant-X70	PYNQ-Z1
Computing power	1352GOPS	160 DSP	31TOPS	400 MHz × 7200 INT8 DSP	125 MHz × 220 DSP
Power Consumption	12 V × 5 A AC Adaptor	12 W	*	*	*
CPU	Cortex-A53 × 4 + Cortex-R5 × 2	Cortex-M1 + Cortex-M3	Cortex-A78AE + Cortex-R52	*	2 × Cortex-A9
GPU	MALI-400MP2	*	Mali-G78AE	*	*
AI accelerator	FPGA AI engine(DPU)	FPGA	AIE-MLv2	FPGA	FPGA
RAM	1/2/4/8 GB DDR4	4 GB DDR3	*	*	512 MB DDR3
Supported DL frameworks	Caffe, Tensorflow, PyTorch, Keras	*	PyTorch, TensorFlow, ONNx	*	*

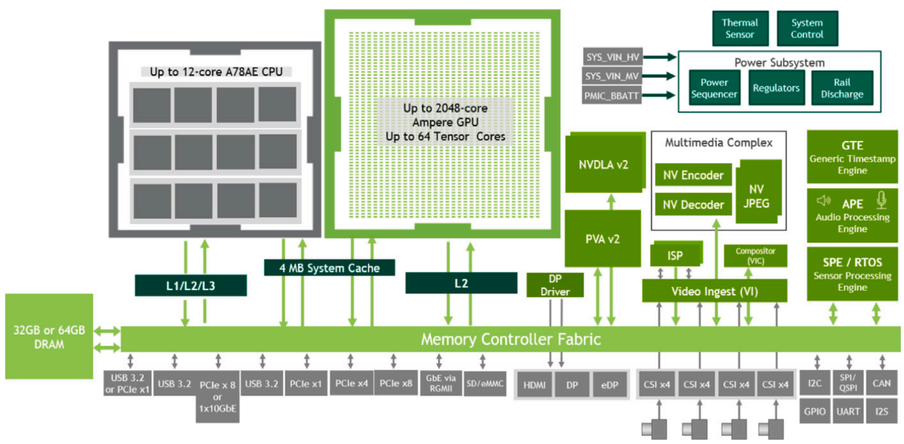

Fig. 3. Jetson AGX ORIN. Provided by [22].

including video encoding and decoding, physical process simulation, etc. The tensor core is optimized for mixed-precision matrix operations. NVDLA2 is designed for convolutional neural networks: product, activation function, pooling, normalization operations, and memory-to-memory transfer operations of tensors are optimized for deep learning acceleration. Other edge devices have their deep-learning acceleration methods, including Google's TPU, Intel's video process unit (VPU), and so on.

As for the FPGA, in Table 5, many studies use FPGA to develop AI models and deploy them in practice [23]. Authors [24] use a Kintex 7 xc7k160t device for emotion recognition of EEG signals, achieving the performance of 128.2 GMACS/W, and uses only 196 KGates (NAND2) with an average power consumption of 0.039 mW in TSMC 90 nm. Authors [25] develop a system that seamlessly integrates the AMD Xilinx FPGA into a customized circuit configuration. They utilize cameras as input sensors to capture road scenes and employ a Deep Learning Processing Unit (DPU) to execute the YOLOv3 model, enhancing the speed and accuracy of defect detection. Authors [26] use iCE40UP5K from Lattice to classify ventricular beat in electrocardiogram achieving accuracy of 97.5%, sensitivity of 85.7%, specificity of 99.0%, precision of 92.3%, and F1-score of 88.9% while consuming only 10.5 μW of dynamic power dissipation. Authors [27] use PYNQ-Z1 to test their FPGA and DNN co-design method and achieve about double frames per second with 40% low power consumption.

In summary, the manufacturing technology of the chip and the number of transistors determine the upper bound of computing power. Hardware accelerators and software acceleration algorithms are proposed to achieve the theoretical upper bound of computing power as closely as possible. Besides, other factors such as memory frequency, bandwidth, cooling structure, and power consumption control will also affect actual performance.

Previously, there have been many precedents for using edge devices and CNNs to solve practical problems: assisting tennis robots in detecting tennis balls [28], helping the cleaning robot Panthera to identify cracks and garbage [29], identifying lanes for automatic driving [30]. These studies focus on solving practical and meaningful problems with CNNs. But they all have limitations: the networks are old, and their performance needs to be improved. However, SOTA models often need strong computing power and large memory size, making it challenging to deploy directly on these AI edge devices. Thus, research on how to get lightweight networks is promising. Currently, mainstream network compression methods include network pruning, network quantization (bit precision reduction), and knowledge distillation. At the same time, designing a lightweight network module has a similar effect. This article will mainly review the methods that can be implemented on general-purpose hardware and are used by other researchers at a high frequency.

Currently, there are four main approaches to obtaining a lightweight CNN network:

- Efficient architectures: This involves designing CNNs with fewer parameters while maintaining the same representation ability.
- Network Pruning: This technique reduces parameters by removing unnecessary connections, convolution kernels, or channels between neurons.
- Model Quantization: Also known as bit precision reduction, this method reduces the model's size and speeds up the calculation by mapping the weight of 32-bit floating-point numbers to 16-bit floating-point numbers, 8-bit integers, etc.
- Knowledge Distillation: Based on the "teacher–student" training method, the knowledge of the large model is "distilled" to the small model so that the small model can perform as close as possible to the large model.

Table 6
Research on Network pruning in edge devices.

Refs	Applications and tasks	Pruning methods	Network	Parameters before/after pruning	Edge devices	Datasets	Performance before/after pruning	Inference time[c] before/after pruning
[31]	Monocular depth estimation	NetAdapt(optimized filter-wise pruning) [32]	MobileNet-based encoder/decoder network	3.93/1.34M	Jetson TX2	NYU Depth v2	RMSE: 0.599/0.604	8.2/5.6 ms
[33]	Ginger detection	Channel-wise and layer-wise pruning based on the BN scaling factor	YOLOV3	234/32.7 MB	Jetson Nano	Custom dataset	mAP: 0.981/0.98	*/20 FPS
[34]	Aerial infrared pedestrian Detection	Channel-wise and layer-wise pruning based on the BN scaling factor	YOLOV3	238.9/10.7 MB	Jetson TX2	Custom dataset	AP: 0.932/0.915	3.7/8 FPS
[35]	Digital signal modulation recognition	Filter-wise pruning based on APoZ	AlexNet	60/2M	Jetson TX2	CSI	*[a]	19/6 ms
[36]	Monocular Road Segmentation	Channel pruning based on the BN scaling factor	Resnet18-based encoder/decoder network	*/0.16M	Jetson TX2	KITTI-ROAD CamVid Cityscapes	mIoU */0.9614 */0.9611 s */0.957	45/81 FPS
[37]	3D point-clouds classification	Unstructured pruning	3D volumetric network	12/9M[b]	NCS2	ModelNet10	Accuracy: */93.5%	11 ms
[38]	Monocular depth estimation	Channel-wise pruning based on TD3[39] reinforcement learning for	EdgeNet	3.95/1.7M	Jetson TX2 Jetson Nano	NYU-Depth-v2	RMSE 0.562/0.562	19.5/11.4 ms 48.6/27.6 ms
[40]	Vehicle detection in aerial imagery	Filter-wise pruning based on L1 norm	UAV-Net	*/0.10M	Jetson TX2	DLR3K VEDAI	AP: 0.972/0.913 Accuracy: 95.7%/93.5%	15.9/38.2 FPS
[41]	Road type recognition	Cross-layer manifold invariance based pruning	custom CNN	0.19/0.11M	Jetson Nano	Custom dataset	Accuracy: 90.6%/87.4%	*/26 FPS
[42]	Vehicle detection	Filter-wise pruning based on the BN scaling factor	YOLOV3-tiny	40/9.7M	NCS2	UA-DETRAC	mAP: 0.332/0.530	*/46 FPS
[43]	Wildfire detection	Channel-wise pruning based on Fourier transform	MobileNet-V2	*	Jetson Nano	BoWFire	Accuracy: */93.36%	*
[44]	Driver behavior identification	Filter-wise pruning based on L1 norm	DeepConv GRU	7.91/1.68 MB	Jetson Nano Jetson TX2 Xavier	Security driving	Accuracy: 98.4%/97.0%	2.58/2.16 ms 1.17/0.99 ms 0.50/0.45 ms
[45]	pedestrain location in coal mine	Channel-wise pruning based on the BN scaling factor	YOLOV3	236/86.4 MB	Jetson TX2	Coal mine pedestrian	mAP: 93.7%/91.7%	6/16 FPS
[46]	Privacy-Preserving inference on Mobile phone	kernel-pattern pruning based on Frobenius norm	ResNet18	*	Samsung Galaxy S10	ImageNet	Accuracy: 94.1%/94.2%	*/33 ms
[47]	3D point cloud object detection and lane detection	ADMM-based weight pruning	PointPillars Ultra-Fast-Lane-Detection	*	Jetson TX2	KITTI TuSimple	mAP: 78.5/77.1 Accuracy: 95.8%/94.46%	569.56/274.06 ms 67.34/29.46 ms
[48]	Semantic Segmentation	Channel pruning based on soft masks	model based on NAS	3.7/1.3M	Samsung Galaxy S21	Cityscapes	mIoU: 74.7/71.5	30.8/52.6 FPS

[a] The author does not provide a specific value, only mentioning that the accuracy rate decreases by 0.2%–1.2% under different SNRs.
[b] Strictly speaking, unstructured pruning does not reduce the number of parameters, by which the author means the number of parameters whose weight is not zero.
[c] Units: frame per second(FPS) or millisecond per sample (ms).

These methods are not mutually exclusive, and many studies combine multiple lightweight techniques to achieve optimal results.

3. Efficient architectures

Efficient CNN architectures incorporate efficiency and accuracy metrics into the initial architectural design. Early neural networks include the single-layer perceptron [49] and the multi-layer perceptron [50]. However, the performance was extremely limited [51,52]. To overcome the computational bottleneck, at the early stage of deep learning [53], the mainstream networks (e.g., AlexNet [5], VggNet [54], GoogLeNet [55], Densenet [56]) were continuously draining hardware performance to get better results. Although these networks can show good performance, they are limited by hardware's computing power and memory and are challenging to apply in practical environments. Before going into the details of these efficient CNN architectures, we need to understand how to quantify the scale of a neural network.

3.1. Metrics for evaluating the scale of neural networks

There are several metrics widely used to evaluate the scale of neural networks: parameters, computational complexity, memory usage, and memory access cost.

Parameters generally refer to the total count of all learnable weights and biases in the network, which determine the network's complexity and learning capacity. The majority of parameters in CNNs are

Table 7
Research on network quantization in edge devices.

Refs	Applications and tasks	Quantization methods	Network	Parameters before/after quantization	Edge devices	Datasets	Performance before/after quantization	Inference time[a] after quantization
[57]	Medical image classification	8-bit integers quantization	Densenet	27.9/8.4 MB	Jetson Nano Coral dev board	NIH-14 Chest-Xray	AUC: 0.81/0.77	410 ms 24 ms
[58]	Face-mask detection	8-bit integers quantization	Maskdetect	11.5/*/0.98 MB	NCS Coral USB	Custom dataset	Accuracy: 94.2%/95.0%	18 FPS 19 FPS
[59]	Vineyard trunk detection	8-bit integers quantization	SSD MobilenetV2	*	Coral USB	Custom dataset	AP: 0.53	23.14 ms
[60]	Tree delineation segmentation for UAV	8-bit integers quantization	U-Net	*	Coral dev board	Custom dataset	Accuracy: 89/88%	28 ms
[61]	Dish detection for empty-dish recycling robots	16-bit floats quantization	YOLOV2-Tiny	*/11.7M	Jetson Nano	Custom dataset	mAP: 0.9738/0.974	5/30.5 FPS
[62]	Vehicle detection	16-bit floats quantization	YOLOV3-Tiny	*/8.7M	Jetson TX2	KITTI	mAP: */81.9	29.1/52.4 FPS
[63]	Workpiece defect detection	8-bit integers quantization	InceptionV3	*	Jetson Xavier NX	Custom dataset	Accuracy: 97.7%/83.1%	29.08 FPS
[64]	Video comprehension	8-bit integers quantization	DEEPEYE	*/18.8 MB	ARM-core board	UCF11	Accuracy: */86.09%	*/24 FPS
[65]	Object Detection	8-Bit Fixed-Point Quantization	RFA-YOLO	*/24.75M	FPGA	DIOR	mAP */64.85	*/27.97 FPS
[66]	Single-Image Super Resolution	8-bit integers quantization	XLSR	*/22K	Samsung Galaxy A21s	Div2K	SSIM: */51.02	*/44.85 ms
[67]	grape bunch detection	8-bit integers quantization	SSD MobileNetV1	*	Coral USB Accelerator	custom dataset	mAP: */55.78	*/93.12 ms
[68]	video compression	8-bit integers quantization	MobileNVC	*/12.42M	Snapdragon 8 Gen 2	HEVC-B	PSNR: 32.5/124.9	*/11 ms
[69]	image classification	activations quantized to 1 bit, and weights quantized to 2 bits	ResNet18	*	Raspberry 4B	VWW	Accuracy: 93.54/91.45%	274/45.8 ms
[70]	image classification	4-bit integers quantization	ResNet18	*	XC7Z045	ImageNet	Accuracy: 69.76/70.27%	*/99.1 FPS

[a] Units: frame per second (FPS) or millisecond per sample (ms).

found in the convolutional and fully connected layers. Other network layers (s.a., pooling layer, activation layer, BN layer) have few or no parameters. Many CNNs and their variants' parameters focus on the convolutional layer, while the Transformer and its variant (e.g., ViT) focus almost entirely on the fully connected layer. Therefore, most lightweight work based on CNNs focuses on how to reduce the parameters in the convolutional layer. In contrast, the work based on the ViT series focuses on how to reduce the parameters in the fully connected layer.

The parameters are only related to the network structure, not the input size. The computational complexity of the convolutional layer is related to both. It refers to the sum of the number of addition and multiplication operations when a network performs a complete inference on the input data. Since the computation of multiplication is more complex than that of addition, researchers generally ignore the addition operations and only focus on the multiplication operations. Meanwhile, since the mainstream deep neural networks use the Float32 data type, researchers tend to use FLOPs (floating-point operations) to represent the computational complexity of CNNs.

Memory usage is another metric for practical deployment. It refers to the byte count necessary for storing weights, inputs, outputs, and intermediate features during neural network training and inference. Generally, training networks require a larger batch size because memory usage for intermediate features increases dramatically. However, AI edge devices generally need to infer the network instead of network training, so they do not need to store gradients. The memory of the Jetson Nano is 2/4 GB, and the memory of the Coral Dev Board is only 1 GB. In fact, the training graphics card has a separate memory, which is independent of the memory occupied by CPU processes. However, the CPU and GPU (or VPN, TPU) in edge devices need to share the same memory, so even if the network input batch size is set to small, the memory of most low-cost edge devices is still insufficient.

After researchers deploy their deep neural networks in AI edge devices, another practical problem arises: Why does neural network inference time actually consume much longer than $\frac{FLOPs}{FLOPS}$? $FLOPs$ is the computational complexity, while $FLOPS$ is the computing power of the device. Memory access cost is one of the main reasons. We have mentioned that when the network runs, the computer stores the network weights and data in memory, but the computation process does not take place in memory. To get the calculation result, the computer needs memory read/write operations and calculation operations. In AI edge devices, the speed of memory reading and writing is significantly slower than the speed of computation. The whole computation process requires a huge amount of memory for reading and writing operations. This is why the actual inference speed of deep neural networks is much slower than $\frac{FLOPs}{FLOPS}$. The methods of calculating different metrics on different modules are listed in Table 11.

3.2. Analysis and comparison of efficient CNN architectures

After understanding how to measure the scale of a deep neural network, this subsection aims to analyze the current mainstream efficient CNN architectures. Some CNN architectures are shown in Table 12. These networks are mainly applied in the field of computer vision (mainly CNNs) but also include ViT [105], which has become popular in the recent years. The vision requirement at the edge is extremely large: face recognition, autonomous driving [106], defect detection of fabric and metal devices [107], robot localization, recognition [108], etc. At the same time, the networks being used in the visual field are generally large and difficult to deploy directly on edge devices, thus the

Table 8
Research on knowledge distillation in edge devices.

Refs	Applications and tasks	KD methods	Teacher model and performance	Student model and performance	Edge devices	Datasets	Inference time[e]
[71]	Monocular depth estimation	Feature distillation	ResNeSt101 RMSE: 3.545	MobileNet RMSE: 3.951	Snapdragon 888 Kirin 970	MAI2021	46.9 ms 76.8 ms
[72]	Optical flow estimation	Soft label distillation	FlowNet-CSS * FlowNet-S * DispNet-CSS *	DWARF EPE: 6.44	Jetson TX2	Train: KITTI Test: FlyingThings 3D	1.59s
[73]	Driver drowsiness detection	Soft label distillation	VGGnet19 Accuracy: 92.5%	VGGnet16 Accuracy: 95.0%	Jetson TX2	ZJU	*
[74]	Pose estimation and Localization for pose estimates	Soft label distillation	NetVLAD *	MobileNetVLAD Median error: 0.029 m	Jetson TX2 [c]	Google Landmarks NCLT	55 ms
[75]	Monocular depth estimation	Soft label distillation	MiDaS *	RMSE MonoDepth2 6.000 PyDNet 6.138 DSNet 5.977 FastDepth 6.017[d]	iPhone XS	KITTI TUM NYU	9.94 FPS 11.05 FPS 58.86 FPS 50.31 FPS
[76]	Fabric defect detection	Feature distillation	YOLOV5 mAP: 0.447	Simplified YOLOV5 mAP: 0.406	Jetson TX2	Xuelang Tianchi AI Challenge	16 ms
[77]	On-road risk detection	Soft label distillation	ResNet20 Accuracy: 98.67%	3-layer CNN Accuracy: 96.35%	NCS2	Custom dataset	53 FPS
[78]	Multi-person action recognition	Soft label distillation	R3D-18 *	R3D-10 Recall: 91.26%	NCS2	Custom dataset	0.36 s
[79]	Face verification and recognition	Soft label distillation	Dlib-Resnet-v1 Accuracy: 99.18%	DenseNet121 Accuracy: 98.52% DenseNet+STN Accuracy: 98.52%	NCS2	train: CASIA test: LFW	67 ms
[80]	Low illumination image enhancement	Feature distillation	U-net SSIM: 0.534 PSNR: 13.62	DwG SSIM: 0.508 PSNR: 12.82	Jetson Xavier NX	BDD100k	20 ms.
[81]	Object detection	Soft label distillation	VGG16+Faster R-CNN mAP: 68.0	Pruned VGG16+Faster R-CNN mAP:59.4	Jetson Nano	PASCAL VOC 2007	78.8 ms
[82]	Object detection	Soft label distillation	YOLOv3-M mAP: 78.92%	YOLOv3-Ms mAP:78.54%	PYNQ-Z1	PASCAL VOC	0.56 s
[83,84]	Face Detection	Feature distillation	Resnet50-SSH Easy Set: 0.955 Medium Set :0.940 Hard Set: 0.844	MobileNet+Tiny-SSH Easy Set: 0.921 Medium Set: 0.899 Hard Set :0.817	Raspberry PI 3B+	Wider Face	16.26 ms
[85]	Medical Diagnosis	Feature distillation	Resnet50 malignant : 0.9835 benign : 0.9794	Resnet18 malignant : 0.9771 benign : 0.9768	Samsung Galaxy S3 S6 S8	malignant benign	*
[86]	Vehicle Detection	Feature distillation	*	Tiny YOLO mAP COCO:0.28 VOC2012:0.41 IITM-Hetra:0.39	Jetson Nano	COCO VOC2012 IITM-Hetra	*

[a] Their Resnet50 has been compressed [87].
[b] Their student model has been pruned and compressed.
[c] The network inference time, not including the time of SIFT, 2D-3D matching, and other localized algorithms.
[d] [75] on the KITTI dataset.
[e] Units: frame per second (FPS) or millisecond per sample (ms).

lightweight deployment in the visual field is more realistic and practical than the machine translation field [109]. These lightweight approaches are diverse, but the main idea is similar: reduce invalid connections while retaining valid connections.

The Inception module in GoogleNet [55] extracts multi-level features more efficiently by concatenating convolutional kernels of different sizes. The Fire module of SqueezeNet [110] is similar to the Inception module: the original $k \times k$ kernels are replaced by combinations of 1×1 descending convolution and $k \times k$ ascending convolution to reduce parameters. Authors [111] proposed factorizing convolution. They argue that several small 3×1 and 1×3 convolutional kernels can replace a large one. It reduces parameters while retaining the original perceptual field with stronger nonlinearity. MobileNet [112] uses depthwise separable convolution to reduce parameters. The depthwise separable convolution consists of a depthwise convolution and a pointwise convolution. The number of output channels in the depthwise convolution must be the same as the number of input channels, and each output channel is connected to the corresponding input channel only. Densenet [56] consists of dense connections and feature reuse. The input of the ith convolutional layer in DenseBlock is the combination of the 1st, 2nd, … (i-1)th convolutional layers' output. It allows the network to process both large-scale and small-scale information by merging shallow and deep features. In order to solve the heavy computation complexity and large memory usage of DenseNet,

Table 9
Utilizing multiple model compression methods on edge devices.

Refs	Year	Applications and tasks	P	Q	KD	Network	Parameters before/after compression	Edge devices	Datasets	Performance before/after compression	Inference time[a]
[88]	2022	Ship detection	✓	✓		Yolov5-x	92/47 MB	Jetson Nano Xavier	Custom dataset	ACC:0.940/0.963	0.3/5.2 FPS 6.4/19.1 FPS
[89]	2021	Environmental sound recognition	✓	✓		CNN1D	102/36K	Coral Dev Board	BDLib ESC-10 ESC-50 UrbanSound	74.4%/63.89% 83.3%/78.5% 63.1%/43.5% 60.0%/56.3%	*/1.43 ms
[90]	2022	Human detection in marine environment	✓	✓		YoloV4	63.9/1.02M	KV260 Movidius VPU	swim-mers [91]	mAP:0.7/0.721	11/69 FPS */29.8 FPS
[92]	2021	offensive words spotting	✓	✓		OWSNet	*/66K	Jetson Nano	Google Speech Commands dataset	Acc:*/93.7%	*/0.8 ms
[93]	2020	UAV fault detection	✓	✓		LSTM	13/12K	Airborne ECP	Custom dataset	Acc: */98.6%	262/2.59 ms
[94]	2022	Optical channel equalizer	✓	✓		MLP	201/26KB	Jetson Nano	Custom dataset	*	37.2/16.2 ms
[95]	2021	Forest fire detection	✓		✓	YOLOV4-MobileNet	63.9/2.63M	Jetson Xavier	Custom dataset	mAP:0.67/0.631	153.8/37.4 ms
[96]	2020	Survivor detection in UAV thermal imagery	✓		✓	Yolov3-Mobilenet	374/22.7 MB	Jetson TX2	Custom dataset	mAP:0.85/0.62	8.45/26.6 FPS
[87]	2021	Head-pose estimation	✓		✓	Resnet50	*	Jetson Nano Jetson Xavier	AFLW2000	MAE:*/5.25	220/8 ms 30/5 ms
[97]	2023	semantic segmentation		✓	✓	AutoSegEdge	*/2.35M	Jetson NX	Cityscapes	mIoU:*/70.3%	*/16.6 FPS
[98]	2023	general computer vision tasks	✓	✓	✓	Swin-Base ViT	87.90/6.88M	AGX orin	COCO	Top-1 Acc(%):84.5/84.4	140/369 fps
[99]	2023	fault diagnosis		✓	✓	TFAM1DCNN	547/171KB	ARM Cortex-M4	Bearing	Acc:*/97.2%	*/200 ms
[100]	2020	image generation	✓		✓	CycleGAN	11.3/0.34M	Jetson Nano Jetson Xavier	horse to zebra	FID 61.53/64.95	*/0.16s */0.026 s
[101]	2023	Single-Image Super-Resolution		✓	✓	EdgeSRGAN	1.5/0.66M	Coral USB Accelerator	Set5 Set14 BSD100 Manga109 Urban100	SSIM:0.842/0.837 0.701/0.715 0.648/0.644 0.86/0.855 0.728/0.716	0.48/2.66 FPS
[102]	2023	remote sensing recognition	✓	✓		MobileNetV1	82/8.2 MB	ZCU104	SLOC4 WHU-RS19 RSSCN7 OPT-Aircraft v1.0	*	*
[103]	2019	object detection	✓	✓		HX-LPNet	*	TDA2PX	BDD100K	mAP:*/52.0	*/22.47 FPS
[104]	2022	object detection	✓	✓	✓	Yolov4-CSP	200.51/24.26 MB	ZCU104	MS COCO	mAP:0.653/0.627	*/33.34 FPS

[a] P: Pruning. Q: Quantization. KD: Knowledge Distillation.

ConDenseNet [113] introduces group convolutions based on dense connection. Connections only exist between input and output channels within the same group. Neural Architecture Search (NAS [114]) aims to automatically design neural networks. Other efficient CNN architectures incorporate attention mechanisms [115] to further reduce the computational cost.

3.3. Summary

These efficient networks and modules in Table 12 are highly representative in practical applications. These networks are widely applied in classification, object detection, and semantic segmentation tasks. Most utilized or improved lightweight networks are designed from them. Their main ideas are similar: reduce invalid connections while retaining valid connections.

In practice, choosing the efficient CNN architecture depends on the specific requirements of the task at hand, such as accuracy and the available computational resources. We find that the majority of previous research has focused on low parameters, and low computation complexity, ignoring memory usage, and memory access cost. For example, the depthwise separable convolution achieves nearly an 88% reduction in parameters, but increases about 5 times memory access cost. This results in a slower inference speed on edge devices. Thus, when designing efficient CNN architectures, it is necessary to balance four metrics for practical applications.

Recently, transformers have also seen lightweight variants, such as Mobile-Former [116], which runs MobileNet and Transformer in parallel, and LeViT [117], which incorporates CNN feature maps and quickly reduces feature map size for acceleration. With the development of ViT series, many researchers were amazed by the high performance but troubled by the large parameters brought by fully connected layers, which are challenging to be accelerated by existing hardware. So many lightweight versions of Transformer were proposed [116,117]. However, since there is currently limited acceleration for the fully connected layer in AI edge devices, while ViT series requires large amounts of data to guarantee performance.

4. Network pruning

The connections in neural networks are very dense, but not all connections are effective. Network pruning was proposed [118–120] in the last century. The main ideas contained in these old algorithms are direct: 1. Compare the network performance before and after removing

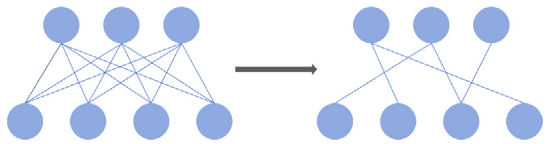

Fig. 4. Element-wise pruning.

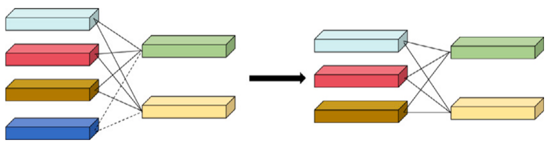

Fig. 5. Filter-wise pruning.

weights and prune those redundant ones. 2. Design a particular magnitude to calculate the importance of weights and prune weights whose importance is lower than a threshold.

Network pruning can be divided into structured and unstructured pruning according to whether the connection is actually removed. According to the pruning granularity, it can be divided into element-wise pruning, filter-wise pruning, channel-wise pruning, and layer-wise pruning [121]. We will introduce these pruning methods in the following sections. **We also collect research on applying network pruning to AI edge devices and make Table 6 for comparison in detail.**

4.1. Element-wise pruning

Among different kinds of pruning, element-wise pruning appeared earliest [118–120]. Because the early neural network structures were usually simple MLP, element-wise pruning can remove those useless connections in Fig. 4. The blue cells represent neurons with a weight value; the lines represent the connections between neurons in two adjacent layers; element-wise pruning can remove the redundant connections among them. It significantly reduces both parameters and computing complexity.

However, element-wise pruning for convolutional layers has not yet been implemented well. It only sets the value of useless weights to 0, and this operation does not reduce the four metrics (parameters, computational complexity, memory usage, and memory access cost) of the network we mentioned [122–124]. In order to accelerate the convolutional neural network with element-wise pruning, software methods such as building an index table [125] and hardware for sparse matrix acceleration must be used. However, the work in this field is not mature. Currently, the most widely used pruning methods for CNNs are still channel-wise and filter-wise pruning.

Although element-wise pruning is currently not applicable to CNNs, it is suitable for fully connected layers. ViT can benefit from element-wise pruning. Research on applying structured element-wise pruning [126] for ViT has appeared.

4.2. Channel-wise pruning, filter-wise pruning

Channel-wise pruning and filter-wise pruning can be easily implemented by removing useless convolution kernel groups and corresponding convolution kernels of the next layer. The implementation methods are the same as Fig. 5. These blocks represent distinct convolutional filters. Filter-wise pruning measures and removes unimportant filters along with their corresponding connections. The difference between them is that when judging the importance of the convolution kernel group, the former is based on the channel information extracted by the convolution kernel groups, While the latter is based on the weights of the convolution kernel group itself.

For example, a study [127] uses the L1 norm as the magnitude for measuring the importance of the convolution kernel group. The principle is also very intuitive: if the L1 norm of a convolution kernel group is minimal, this convolution kernel group is not essential. a study [128] uses the similarity between convolution kernel groups as the magnitude. The two studies are based on the convolution kernel's weights, which do not require data as input. Authors [129] chose the average percentage of zeros (APoZ) output by the activation layer when calculating the data as the magnitude. Authors [130] regarded the scaling factor in the BN layer as the magnitude. Researchers also took entropy [131], the impact on the loss function when inputting data [132,133], the rank of the convolution kernel matrix [134] as magnitude. As for the pruning methods based on the reconstruction error, they remove weights based on the output error before and after removing particular weights. The selection methods [135–137] include the greedy method, Lasso regression, etc.

Instead of manually designing magnitude, some researchers proposed automatic pruning [138,139] based on reinforcement learning, which brought revolutionary progress to channel (filter)-wise pruning.

In early research, fixed pruning ratio were applied on every convolutional layer. However, some researchers found that flexible-rate pruning performs better because different convolutional layers have different sensitivities [140]. Layers with greater depth and width may retain their original performance after a high ratio of pruning, while shallow and narrow layers may retain their original performance after a low ratio of pruning.

Although channel (filter)-wise pruning is widely used, it faces a problem: some useful weights may be pruned with other useless weights together. In order to prevent this from happening, some researchers have introduced sparse training [141,142], which makes the network sparse and the effective weights become concentrated, facilitating more efficient pruning.

4.3. Layer-wise pruning

Compared with channel (filter)-wise pruning, layer-wise pruning has a larger granularity. It is proposed mainly for pruning the shortcuts in ResNet and similar structures. Although shortcuts and concatenation can provide multi-scale information or protect the information in deep layers, sometimes the information brought by these structures is redundant. At this time, it is necessary to prune these structures and their convolutional layers to reduce parameters. A typical layer-wise pruning method could be found in [143].

Compared with channel (filter)-wise pruning, layer-wise pruning can more thoroughly remove useless shortcut branches and save a lot of computing resources. The effect is particularly significant in multi-branch networks like YOLOV3 [144]. So far, layer-wise pruning is mainly used on the shortcut branch, resulting in that it cannot be used as the primary pruning method. Combining channel (filter)-wise and layer-wise pruning is a good choice in the future.

4.4. Other kinds of network pruning

There are many other classification bases for network pruning. For example, according to the relationship between pruning and model, pruning methods can also be divided into pruning before, during and after training. Pruning after training is the most commonly used method [127,130,145]. It is stable and simple. The disadvantage of this method is that the pruning and training processes in the iterative process are time-consuming.

Based on pruning after training, some researchers proposed to remove the fine-tuning process and directly prune the network [146] in the training process using a standard learning rate. This method can significantly reduce the overall time cost. The disadvantage is that the original performance may not be restored during model training and pruning.

In the above two methods, even if the network is pruned into a small subnetwork at the end, the cost of training a large network is still large during the pruning process. To address this, [147] proposed the lottery ticket hypothesis: in a densely connected, randomly initialized DNN, a subnetwork exists that can perform as well as the original network after enough training epochs. This subnetwork is called the 'lottery ticket'. But how to find this 'lottery ticket'? At present, the SOTA pre-training pruning methods are SNIP [133], GraSP [148], Synflow [149], applying element-wise pruning method based on magnitudes to prune before training. At the same time, although pre-training pruning methods can effectively reduce the cost of training, most existing SOTA pre-training pruning methods obtain a worse network than post-training pruning [150].

Based on filter-wise pruning, some researchers proposed cluster pruning [151]. Cluster pruning method first ranks filters within each layer based on their importance and groups filters into clusters. Then it prunes clusters one by one based on the average importance of the filters, until the desired trade-off between accuracy and pruning objective is achieved.

In addition, pruning can also be divided into soft/hard pruning. Hard pruning is what we discussed before. Soft pruning only sets the values of unimportant convolution kernel groups to 0, and these convolution kernel groups will still participate in training and gradient feedback in the following training epoch. Until the entire training process is over, the convolution kernel groups that are still 0 will be pruned. Compared with hard pruning, soft pruning has larger model capacity and less dependence on the pretrained model. However, as a trade-off, soft pruning needs more computational resources than hard pruning [152].

4.5. Network pruning applied to edge

Some representative studies applying network pruning to AI edge devices are listed in Table 6. We have the following conclusions:

1. Network pruning has been widely used in many fields, including object detection, semantic segmentation, depth estimation, and point-cloud classification.

2. It is clear from these studies that network pruning is an effective technique for reducing the size of deep models while maintaining their performance. For example, [34] compressed YOLOV3 from 238.9 MB to 10.7 MB, and the pruning rate is 95.5%. At such a high pruning rate, the model's mAP only drops from 0.932 to 0.915. However, it is worth noting that over-pruning may lead to severe accuracy loss. Therefore, researchers need to carefully apply pruning methods and set pruning rates before deploying them to AI edge devices.

3. Most studies in Table 6 choose to apply channel(filter)-wise pruning. Because channel(filter)-wise pruning is simple to implement and has proven effective in experiments, and layer-wise pruning is only suitable for networks with shortcut branches, unstructured pruning makes it hard to speed up the network inference.

It is worth mentioning that, at present, automatic channel/filter-wise pruning based on reinforcement learning [39] or other strategies showed excellent performance. However, it requires significant time and computing power. As a result, many researchers tend to use traditional manual pruning methods. Apart from automatic pruning, element-wise structured pruning is also promising. It is a fantastic pruning method because it can remove most invalid connections accurately. However, it is challenging to implement: we need to design a new data structure to present pruned convolutional kernels and design hardware support to accelerate its calculation. Alternatively, we can design hardware that supports sparse matrix acceleration. Both of them are worth further studying.

5. Network quantization

The concept of quantization comes from digital signals and refers to the process of approximating continuous values of signals to a finite number of discrete values. The data types in the computer can be distinguished according to the quantization precision and quantization range, such as the commonly used 16-bit integer number (Int16), whose value range is an integer between [−32768, 32767]. In deep learning models, we generally use 32-bit single-precision floating-point numbers (Float32), which can express the range of $\pm[1.18 \times 10^{-38}, 3.4 \times 10 \times 10^{38}]$ [153], to store their weights. Network quantization quantizes the Float32 in the model to a lower number of digits to compress the model.

We will introduce mainstream network quantization methods in the following sections. **And we collect research on applying network quantization to AI edge devices and make Table 7 for comparison in detail.**

5.1. Classic 8-bit quantization

Network quantization is different from simply quantizing a network's weights, input, and output to a lower number of bits. The earliest network quantization appeared in 2011 [154]. The author proposed quantizing the activation value to an 8-bit unsigned char and the value of the weights to an 8-bit signed char. This approach effectively reduces memory usage and speeds up the calculation. Moreover, the accuracy loss reaches a small value. Authors [155] proposed compressing the 32-bit gradient and activation values to 8-bit floating point numbers. Authors [156] first trained the network with Float32 and then used 16-bit fixed-point calculations for retraining after the training was completed. Besides, researchers also proposed many different quantization methods in the early years. A representative example is Song Han's [125], whose research first clusters the weight values and then represents the original complete weights by index and values of clustering centers. Among these studies, the Int8 (8-bit integers) quantization [157] proposed by Google engineers has become the mainstream in the industry. It also lays the foundation for the subsequent mainstream network quantization methods. Under Int8 network quantization, the training process is still under Float32. Int8 is used to 'simulate' Float32 operations during inference to retain the model's performance as much as possible.

5.2. Quantization with different bits

In practice, there is 16-bit quantization for better performance and 4-bit, 2-bit, or even 1-bit binary quantization for further model compression.

5.2.1. 16-Bit quantization

16-bit quantization is divided into two routes. One is to use Int16 to simulate Float32 calculation [156,158], and the other is to use half-precision 16-bit floating-point numbers to replace Float32 in the network directly [159,160]. Compared with Float32 and Int8, 16-bit quantization is a compromise choice. However, since most hardware does not support the acceleration of 16-bit fixed-point operations, the studies that apply Int16 quantization are less than Int8 quantization. The quantization using Float16 is mainly used for mixed-precision training to speed up the training process of neural networks [159,160]. Float16 quantization can also be used to accelerate the inference process. However, only specific hardware devices can support and accelerate Float16 operations, such as Nvidia's Jetson series.

5.2.2. Lower-bit quantization

There are various quantization methods below 8 bits, including 4-bit quantization [161–163], 3-bit quantization [164], 2-bit quantization [165,166], binary network [167–169], ternary network [170,171]. These studies have achieved a very high compression ratio while retaining as much performance as possible in the original network. Even under some special conditions, the quantized network performs better than the pre-quantized network [162,164]. However: 1. Currently, most AI edge devices are based on Int8 calculations and accelerated for Int8 calculations. If quantization with lower bits is used, it may not perform better because it is not accelerated by hardware support. 2. Network quantization can effectively reduce overfitting, which has been proved in practice [162,164]. The network deployed in AI edge devices often has a relatively simple structure and shallow layers, which is prone to underfitting. Thus, low-bit quantization may lead to a severe loss of performance.

5.3. Post-training quantization and quantization-aware training

Based on whether the quantization operation is involved in training, quantization can also be divided into post-training quantization (PTQ) [155,156] and quantization-aware training (QAT) [157,172]. The disadvantage of PTQ is that there are many rounding operations in the quantization process without calibration, which leads to accuracy loss. If PTQ is used for a big model such as ResNet, its robustness could overcome the accuracy loss [162]. However, if PTQ is used for a small model such as MobileNet, there will be a more severe accuracy loss. To address this problem, authors [157] proposed QAT, which allows training quantized models by inserting a Fake Quantization Node in the network. Compared with PTQ, the model obtained by QAT has higher accuracy. Although QAT needs to be retrained after getting the baseline model, it is worth it if the accuracy can be significantly improved.

5.4. Other problems

Many problems also need to be considered when implementing network quantization: when quantizing Float32 to low-bit integers, should we use a linear or nonlinear mapping function? Does each layer share a set of S and Z parameters, or does each channel use a separate set of S and Z parameters in the Appendix? When quantizing the output of the activation layer, do we calculate the S and Z parameters online or using the fixed parameters calculated during training? These problems are not yet conclusive and require further investigation.

It is worth mentioning that, similar to automatic pruning [138], some researchers also propose an automatic quantization framework [173] that uses different bit quantization layer by layer. However, most AI edge devices currently do not support calculation acceleration of less than 8-bit data types. Although automatic quantization is theoretically feasible, the effect in real-world deployment may not be as good as Int8 quantization. The practical implementation of automatic quantification still has a long way to go.

5.5. Network quantization applied to edge

Table 7 compares the related studies that apply network quantization methods to AI edge hardware in practice. The quantization methods used in these studies are from the software toolkits of the corresponding AI edge devices, and few researchers have implemented lower-bit network quantization or improved 8-bit/16-bit quantization on AI edge devices. The reasons may be as follows:

1. Without hardware support, lower-bit quantization may not perform better than 8-bit/16-bit quantization. Because big networks tend to overfit, low-bit quantization could maintain the original performance and mitigate overfitting sometimes. However, most networks deployed on AI edge devices are small networks that tend to underfit, and low-bit quantization can significantly reduce their accuracy. Thus, few researchers have applied lower-bit quantization on AI edge devices. Besides, many researchers proposed improved quantization methods in theory but did not obtain proper hardware support. Thus, few researchers have applied improved 8-bit/16-bit quantization on edge.

2. Typical AI edge devices, like Jetson Nano and NCS2, support Int8 quantization acceleration using their software toolkits. The Jetson series devices also support FP16 quantization acceleration. Quantization methods provided by AI edge devices' toolkits have been improved by engineers many times. Many engineers mainly research further improving Int8 and FP16 quantization. The academic community mainly focused on deep compression methods such as binary network [167] and ternary weight network [170], which have not fully applied in practice.

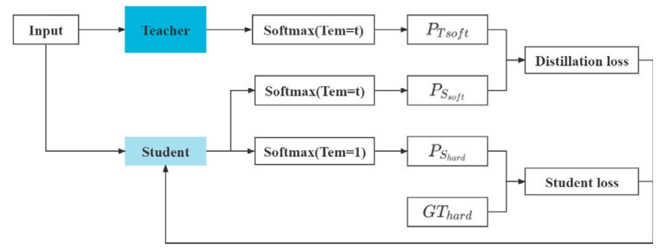

Fig. 6. Basic process of knowledge distillation via labels.

6. Knowledge distillation

The concept of knowledge distillation was first proposed in 2015 [174]. It is a special kind of transfer learning proposed to solve these challenges: 1. it is difficult to deploy big networks in AI edge devices. 2. the performance of small networks is usually not ideal. To address these problems, authors [174] proposed to use the 'teacher–student' method to distil the knowledge learned by the big network to the small network, then achieved the purpose of compressing the model and improving performance. Currently, it has developed numerous different branches: according to the knowledge form, it can be divided into label knowledge, intermediate layer knowledge, and parameter knowledge; according to the distillation method, it can be divided into offline distillation, online distillation, self-distillation, data-free distillation, and multi-model distillation [175,176]. We will introduce existing knowledge distillation methods later. **And we collect research on applying knowledge distillation to AI edge devices and make Table 8 for comparison in detail.**

6.1. Knowledge distillation via labels

Label knowledge is the first kind of knowledge distillation [174]. Its process is shown in Fig. 6. In this process, we need a big network as the 'teacher' and a small network as the 'student'. Through computing the loss between soft-label output by the student and teacher model and then performing gradient feedback, the knowledge is 'distilled' from teacher to student. A typical loss function used in label knowledge distillation is shown in Eq. (1).

$$L_{KD} = H(y_{true}, P_S) + \lambda H(P_T^t, P_S^t), \tag{1}$$

where H is the cross entropy function, y_{true} is the annotations of used data. P_S is the probability distribution output by the student model after the softmax function. P_T^t and P_S^t are the probability distribution output by the teacher and student models at 'temperature t'. The process of how t affects the probability distribution is in Eq. (2).

$$P_i = \frac{\exp\left(z_i/t\right)}{\sum_j \exp\left(z_j/t\right)}. \tag{2}$$

Researchers often set $t > 1$ to make the probability distribution smoother. By smoothing the probability distribution, knowledge distillation can enable the teacher network to teach the student network more comprehensive knowledge. [174].

Knowledge distillation via labels can also be used in object detection. But the performance of directly applying the teacher's bound regression result is not ideal [177]. Then, researchers use intermediate representation (or feature) for knowledge distillation.

6.2. Knowledge distillation via intermediate representation

Label knowledge is hard to transfer when the task is complicated [177]. Researchers use the intermediate layer's output features as the transferred knowledge [178]. A typical distillation process via intermediate representation is shown in Eq. (3).

$$L_{mid} = \|V - Z\|_2^2, \tag{3}$$

where V and Z are the output of the intermediate layers of the teacher and the student network, respectively. $\|\cdot\|$ is the L2 norm. For example, we can design a network that uses ResNet50 as the backbone to extract features. ResNet50, with added fully connected layers, can complete image classification tasks. ResNet50, with added detectors, can complete object detection tasks. The subsequent network completes different tasks through different methods by using features extracted by backbones. These features are the knowledge we need to transfer.

Feature distillation can also be divided into two categories: same-structured distillation and variational-structured distillation. The former is applied when student and teacher models have similar backbones, like ResNet101 and ResNet18 [179,180]. The latter is applied when the feature maps of teacher and student models cannot match. Researchers proposed different ways to overcome the mismatch [181,182]. Generally, same-structured distillation is more stable and simple than variational-structured distillation. But same-structured distillation requires that student and teacher models should have similar backbones, which is not common in practice. In summary, compared with label knowledge, feature knowledge has higher interpretability and is suitable for complicated tasks.

6.3. Knowledge distillation via parameters

After using labels and intermediate features as knowledge, the researchers further proposed: Can the parameters of the network be regarded as knowledge? Parameter distillation can be divided into two categories: mean teacher and module injection. Mean teacher introduces the updated parameters at the last iteration into the current iteration to make the training process stable, which does not focus on model compression [183]. Module injection is more complex: 1. decompose student networks into a bunch of modules; 2. copy n teacher models, and 'inject' decomposed modules into n teacher model at the corresponding position, so every teacher model has one module from student model; n is the number of decomposed modules; 3. train every new teacher model; 4. extract injected modules and combine them into student model [184]. Though module injection has considerable performance, it is not easy to implement. Thus, few researchers have applied parameter distillation to AI edge deployment.

6.4. Knowledge distillation with different distillation methods

Distillation method can be divided into offline distillation, online distillation, self-distillation, data-free distillation, and multi-model distillation [175,176]. Among them, online distillation, self-distillation, and data-free distillation are not proposed for model compression. Offline distillation is the simplest and most common distillation method. During a distillation process in Fig. 6, the teacher model does not update its parameters [174] but updates the student model's parameters. Multi-model distillation introduces multiple teacher models to teach one student model by taking the weighted mean of their distillation loss value [185,186]. This can combine the advantages of multiple teacher models. But multi-model distillation usually consumes more computing resources. Some researchers noticed that if the size of teacher and student models differs greatly, the distillation result will not be ideal. Another study proposes multi-layer teacher distillation to solve this problem (e.g., a ResNet152 teaches a ResNet50, then the ResNet50 teaches a ResNet18) [187].

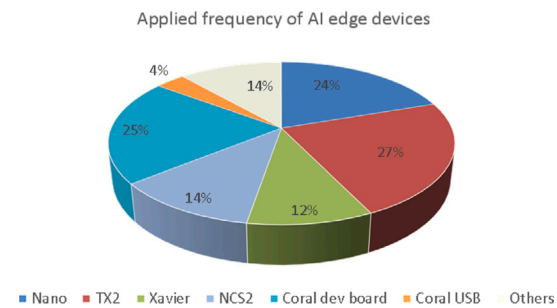

Fig. 7. Applied frequency of AI edge devices.

6.5. Knowledge distillation applied to edge

Table 8 shows the studies on knowledge distillation applied to AI edge devices. We can conclude:

1. Like network pruning, knowledge distillation is widely used in various fields and significantly improves performance. In many studies, student models perform as well as or even better than their teacher models, proving the effectiveness of label and intermediate layer knowledge distillation.
2. Most researchers have implemented label and intermediate layer (feature) knowledge distillation on AI edge devices. Label knowledge distillation is easy to implement and performs well for classification tasks. Intermediate layer distillation is helpful for more complex tasks, such as object detection and semantic segmentation. Due to the difficulty of algorithm implementation, computational cost, and mismatch of the target field, few researchers have applied other knowledge distillation for model compression.

It is worth noting that knowledge distillation can be used not only for model compression but also in cross-modal learning, cross-domain learning, privacy protection, and other fields. Among all knowledge distillation methods, label and intermediate layer distillation are designed and most widely used for model compression.

7. Discussion

In this paper, we have introduced some AI edge devices, lightweight CNNs, and model compression methods, including efficient architecture, network pruning, network quantization, and knowledge distillation. In this section, we will discuss each individual module's advantages, limitations, and relationships and provide some practical suggestions for AI edge deployment.

7.1. AI edge devices

Considering the practical applications and tasks, choosing suitable devices for deployment deserves discussion. Many factors should be considered when choosing the appropriate AI edge device, including the specific tasks, the size and type of models, the deep learning framework, the power consumption and size requirements of the application scenarios, as well as the vitality of the developer community. In this paper, we review about 300 articles. To analyze the research and application status further, we conduct statistics for AI edge devices.

In Fig. 7, the Jetson series is the most popular AI edge device among researchers (Jetson Nano 24%, Jetson TX2 27%, Jetson Xavier 10%), followed by the Coral Dev Board and Coral USB (Coral Dev Board 25%, Coral USB 4%), with the NCS2 being the next most used (NCS2 14%). These three series of AI edge devices account for 86% of all AI edge devices utilized in reviewed articles. The remaining studies employ a variety of other AI edge devices, collectively constituting only 14% of the total. Using the three most popular AI edge device series as examples, we will introduce how to choose suitable devices.

As shown in Table 1, the Jetson series, characterized by its high computational power and big RAM, is the optimal choice for tasks that are complex and demand high real-time performance. Typically, tasks such as object detection and semantic segmentation, which require substantial computational resources, are best served by the Jetson series. Particularly, when tasks need both GPU and CPU for intensive computation, the Jetson's quad-core ARM Cortex-A57 processor provides an additional boost. For simpler tasks with low computational requirements, the NCS2 is an appropriate choice. Because it is more cheap and can support a wider range of deep learning frameworks. If the tasks are complex and power consumption must be strictly controlled, the Coral Dev Board, with the highest power efficiency, is usually chosen. These scenarios represent the typical applications for these three AI edge devices. When task requirements are less stringent, the choice of AI devices often leans more towards personal preference. For example, most researchers tend to apply PyTorch as the DL framework because of its convenience. But Coral Dev Board only support TensorFlow-lite, which is one of the main reasons for its limited usage. In addition, the developer communities for Jetson and NCS2 offer a wealth of practical cases and problem-solving Q&A, which the community for Coral relatively lacks.

In general, these edge devices have distinct advantages and disadvantages: The Jetson series offers the highest computing power and the best CPU and RAM, supports most major deep learning frameworks, and benefits from a mature developer community and numerous practical projects. However, it is the most power-consumption and expensive. The NCS2 consumes the lowest power consumption and cost, but its computing power is weak, and it requires an additional microcomputer to operate. The Coral series excels in the ratio of computing power divided by power consumption and is highly optimized for TensorFlow-Lite, but its limited support for only TensorFlow-Lite can be developer-unfriendly, and its community is less mature compared to Jetson and NCS2. When selecting these devices, it is crucial to balance their strengths against the project's needs and consider if their weaknesses are unacceptable for the application.

7.2. Efficient architectures

Currently, there are a lot of mature studies on how to design/search CNNs [188]. However, it is worth mentioning that lots of studies only focused on parameters and computational complexity. Researchers should be **aware that the memory access cost also significantly affects the deep models' inference time**. When we design a lightweight module or choose a lightweight CNN for deployment, we should comprehensively consider the four metrics: parameters, computational complexity, memory usage, and memory access cost. Each of them is very important in practice.

It is worth mentioning that although the performance of ViT in the computer vision field is impressive [105], there are few studies on deploying lightweight ViT on AI edge devices [189]. ViT's lightweight deployment is worth exploring.

7.3. Network pruning

We can see from Table 6 that existing pruning methods applied on AI edge devices are mainly channel/filter-wise pruning. Because channel/filter-wise pruning is easy to implement by cutting whole convolutional kernel groups. It is easy to achieve a trade-off between performance and parameters by adjusting the pruning rate. At the same time, layer-wise pruning is often used as an auxiliary pruning method in the models with many branches [33,34]. Although the channel/filter-wise pruning based on reinforcement learning is time-consuming, it effectively solves the limitation of manually setting magnitudes [138]. Note: if the existing model is directly pruned, the loss of accuracy will be severe when the pruning ratio is insufficient. The sparse training for the pruned model [141,142] would be helpful in practice [141,142].

In practical deployment, channel/filter-wise pruning has been fully developed. Elements-wise pruning is rarely used because it is challenging to achieve structured elements-wise pruning or sparse matrix acceleration in the convolutional layer. However, with the popularity of ViT, which uses fully connected layers as the main structure, element-wise pruning still has considerable prospects [126]. At the same time, if researchers could implement structured element-wise pruning or accelerate sparse matrix calculation for convolutional layers, it has the potential to prune almost all useless connections precisely.

In the future, network pruning applied to edge devices may focus on the following areas:

1. Manual pruning is still more widely used than automatic channel/filter-wise pruning because automatic pruning requires time and computation resources. Therefore, research on developing an optimized strategy so that automatic pruning may need fewer computation resources is very meaningful.

2. Layer-wise pruning has limitations and can only be used in shortcut branches. However, since big networks do not always perform better than small networks and sometimes most filters in one layer may be removed during manual pruning, it is worth considering whether some middle layers of big networks are necessary.

3. Element-wise pruning has a bright future by removing all invalid connections while maintaining the original performance, but it also faces great challenges in implementation. If someone could implement universal element-wise structured pruning and design AI edge devices that the cloud supports, significant progress would be made in developing network pruning and application of AI edge devices.

7.4. Network quantization

Network quantization is more fundamental to hardware than network pruning. Now, most AI edge devices focus on accelerating Int8 calculation, and only a few devices, like the Jetson series, support accelerating Float16 calculation. Therefore, the famous quantization on AI edge devices is mainly based on Int8 quantization. Meanwhile, because existing deep learning frameworks or software toolkits (TensorRT, OPENVINO, TensorFlow Lite, etc.) have integrated mature quantization implementation methods, it is easy to implement existing 8-bit quantization methods on edge devices. In particular, TensorRT is a platform designed to optimize and accelerate deep learning inference, catering to the high-performance needs of modern AI applications [190]. As an inference optimizer and runtime library, TensorRT ensures AI models run efficiently on NVIDIA GPUs, making it a critical tool for deploying AI models in real-time environments. By performing advanced optimizations such as layer fusion, precision calibration (FP32, FP16, INT8), and kernel auto-tuning, TensorRT can significantly enhance these models' performance, resulting in reduced latency and increased throughput. TensorRT is also Compatible with popular deep learning frameworks like TensorFlow, PyTorch, and ONNX. This flexibility allows developers to easily convert their trained models to TensorRT for enhanced performance. With support for both C++ and Python APIs, TensorRT seamlessly integrates into existing workflows, enabling developers to deploy optimized models across various NVIDIA hardware platforms, from data centers to edge devices.

While quantization is only an auxiliary model compression method, it can significantly compress models. The common Int8 quantization can reduce a typical Float32 model's memory access cost and memory usage by 75%. Because the integer calculation is far simpler than the floating-point calculation, the time consumed for calculation is also significantly reduced. However, it should be noted that if the robustness of the network before quantization is weak, the network's performance after quantization may reduce more. It is another problem for the Int8 quantization to recover the performance of the network before quantization. Besides, how to make AI edge device supporting calculation with lower bits is urgent for dynamic/automatic quantization, which has shown great prospects [173].

In the future, network quantization applied to edge devices may focus on the following areas:

1. Further development of 8-bit and 16-bit quantization. Although the current quantization methods have shown promising results, there is still much to explore regarding details and techniques in Section 5.4.

2. Development of new AI edge devices that support low-bit calculation acceleration. It is not clear whether a big model with low-bit quantization could perform better than a smaller one with 8-bit quantization using the same RAM usage. Currently, most AI edge devices do not support low-bit quantization well, and the big model applying low-bit quantization may have a lower inference speed due to hardware limitations. If low-bit calculations can be accelerated like 8-bit calculations, it will be possible to compare the performance of these two types of quantization more fairly. It is worth mentioning that automatic low-bit quantization [173] also showed considerable performance. However, it also faces the problem of a few AI edge devices supporting accelerating low-bit calculation.

7.5. Knowledge distillation

Compared with the above two model compression methods, knowledge distillation tends to improve the performance of lightweight networks. To further compress a model, we can prune or quantify the student network after knowledge distillation. Label knowledge is most widely used because it is easy to implement by computing loss between the teacher network's output and the student network's output. However, label knowledge is suitable for simple tasks (e.g., classification). For complex tasks like bounding box regression or semantic segmentation, intermediate feature knowledge is more suitable [178]. Parameters knowledge is rarely applied on edge because it involves changing the structure of the teacher–student network, which is hard to implement in practical AI edge devices [184]. Offline distillation is simple and effective. Multi-model distillation can better improve the student network's performance by combining multiple teachers' advantages, but it costs more effort and computing resources.

Like structured network pruning, knowledge distillation is a general model for transferring knowledge between models. Thus, it is not limited by lightweight deployment. Label knowledge suits simple tasks, while intermediate feature knowledge needs teacher and student models that have a similar structure [179,180]. These are many challenges knowledge distillation mainly faces when applied to AI edge devices.

In the future, knowledge distillation applied to edge may focus on the following fields:

1. Although knowledge distillation methods achieve good performance, they lack great interpretability. Many researchers are studying its interpretability to understand how it works [176] and develop better distillation methods based on its essence.

2. Knowledge distillation could be combined more deeply with network pruning: after pruning, we usually retrain the pruned network to recover its performance. Applying knowledge distillation during retraining may further improve the network. More combinations of knowledge distillation and network pruning are needed. These combinations require more computation resources but also could provide better performance.

7.6. Relationship among four lightweight methods

Among the four lightweight methods, efficient architectures can provide a superior backbone, like performing a 'pre-pruning' on big networks. The three model compression methods are independent of each other. Network pruning and quantization accomplish model compression in different ways, while knowledge distillation intends to improve the performance of the compressed model. Researchers can **use the combination of them in one task**(see in Table 9). Some researchers first apply knowledge distillation to improve the performance, then apply network pruning [87,95] or quantization. Some researchers first apply network pruning, then apply network quantization [89,90,92–94]. However, we should be careful when applying the combination of them because it may lead to over-compression and severe loss of accuracy.

Besides, knowledge distillation and network pruning are mainly applied on PCs or servers, while quantization is applied on AI edge devices. Therefore, knowledge distillation and network pruning are usually applied first to obtain a preliminary model; then quantization is applied when the preliminary model is deployed to an AI edge device. These are simple combinations of the three methods. Some research focuses on deeply combining network pruning, network quantization, and knowledge distillation deeper [191–195]. However, their effectiveness and generalization remain to be further verified.

Here we present a straightforward example of how to choose an appropriate edge device and implement lightweight methods. Suppose researchers need to count the persons at a supermarket entrance based on surveillance video. This task requires a high-precision object detection model, and the devices with low computing power, like NCS2, would be chosen. Numerous detection models have been proposed by researchers. After careful selection, the YoloV5 implemented by ultralytics [196] is a suitable choice due to its powerful performance and easy deployment. Since the project is based on PyTorch, the Coral device is not supported, leaving the Jetson Nano as the hardware platform of choice. After testing five different sizes of YoloV5, researchers may find that yolov5-middle has the best accuracy required for the task, but the inference time is slightly longer. Therefore, channel pruning and 8-bit quantization-aware training using the interfaces provided by PyTorch will be conducted. If the accuracy after this process does not meet the task requirements, researchers may need to use yolov5-large as a teacher model for knowledge distillation. Finally, the model is deployed to the Jetson Nano and converted to an 8-bit model using tensorRT for inference.

7.7. Large language models deployment

With the rapid development of AI, large language models (LLMs) have demonstrated exceptional performance in downstream tasks and applications [197]. The model size of LLMs usually exceeds 10 billion (10B) parameters. Some examples are shown in Table 10.

However, in practice, the computational intensity and memory consumption present serious challenges when deploying LLMs on edge devices. These challenges include latency and response time, memory footprint, the balance between accuracy and efficiency, etc. [198]. It results in large computational consumption and carbon emissions [199]. To make full use of LLMs, there is an increasing interest in LLMs' lightweight deployment [200].

Silimar to the standard model compression for general neural networks (e.g., Transformer), model compression for LLMs consists of efficient architectures, pruning, and knowledge distillation [214].

Because many LLMs utilize Transformer, designing efficient Transformer architectures [215] for LLMs has become a hot topic. Some studies reduce the number of model parameters in encoders/decoders to accelerate inference [216,217]. Because self-attention consumes most computation, some studies aim to improve the attention network so as to reduce the computational complexity. Some studies utilize sparse

Table 10
Examples of LLMs' parameters.

Refs	Name	Year	Organization	Parameters
[201]	ChatGPT 3	2020	Open AI	170 B
[202]	MT-NLG	2021	Microsoft & NVIDIA	530 B
[203]	Chinchilla	2022	Deepmind	70 B
[204]	ChatGLM	2023	Tsinghua University	6B-130 B
[205]	Gemini-Nano	2023	Google	3.25 B
[206]	Bloom	2023	Hugging Face	176 B
[207]	Koala	2023	UC Berkeley	13 B
[208]	Alpaca	2023	Stanford University	7 B
[209]	WikiChat	2023	Stanford University	175 B
[210]	LLaVA-1.5	2024	University of Wisconsin–Madison	13 B
[211]	ChatGPT 4o	2024	OpenAI	unknown
[212,213]	LLaMA 3.2	2024	Meta	8B,70 B,405 B

attention [218–222], which limits a token's involvement to a specific subset of tokens determined by a pre-defined sparsity pattern. For example, authors [223] divide the input sequence into several blocks and only perform intra-block attention. Some studies utilize sliding-window attention [224,225], which enables every token to pay attention to neighboring tokens within a sliding window. Authors [226] propose dilated attention, which exponentially reduces attention allocation as the token-to-token distance increases, extending the sequence length to accommodate up to one billion tokens. Authors [227] propose Flash Attention to maximize the utilization of the matrix multiplication capabilities that Tensor Cores excel at within GPUs, while minimizing the proportion of non-matrix multiplication operations. TinyLlama [228] implements Flash Attention and fully sharded data parallel to compress the LLaMA model to 1.1B. Some studies utilize attention sharing/parallel mechasim [229–231] to improve attention computation efficiency. Some studies utilize multi-query attention to reduce computation [232–235]. Multi-query attention is essentially multi-head attention, with the exception that all the different heads share a common set of keys and values. An interesting study proposes a key–value (KV) cache eviction policy (H2O) [236], which chooses some crucial tokens that could significantly impact subsequent decoding processes and preserve their KV cache. SwiftInfer [237] combines StreamingLLM [238] and TensorRT, proposing and implementing attention sink (keeping the KV of initial tokens). Some studies utilize low-rank factorization to speed up [239]. Authors propose Linformer, using a low-rank approximation to reduce the attention to the linear complexity [240]. Authors [241] train a meta early exit classifier to dynamically decide when to stop allocating computational effort. Authors [242] present linear transformers, using linearized attention and causal masking to reduce the complexity.

Existing LLM pruning can be divided into unstructured pruning and structured pruning. Unstructured pruning method s [243–247] focus on individual weights or neurons within the LLM, typically by applying a threshold to nullify parameters that fall below it, to achieve sparse LLMs. Due to retraining LLMs is time-consumption and expensive, authors [248] prune LLMs to achieve at least 50% sparsity in one-shot, without any retraining. Authors [249] propose a new pruning metric, where each weight is evaluated by multiplying its magnitude with the norm of the related input activations. Authors [250,251] focus on retaining core functionalities and employ contextual pruning to reduce model parameters. Structured pruning methods [252–255] usually eliminate complete structural components, including neurons, channels, or entire layers. Authors [256] propose targeted structured pruning, which prunes an LLM to a specified predefined target architecture. Compresso [257] presents a memory-efficient pruning method to reduce parameters, merging Low-Rank Adaptation (LoRA) and L0 regularization.

LLMs quantization techniques are generally divided into two main categories: quantization-aware training (QAT) and post-training quantization (PTQ). QAT involves updating the quantized weights through backpropagation during the training process. QAT is challenging to implement for LLMs because the training process itself is technically complex and demands significant computational resources. Besides, QAT requires access to training data, which is difficult for LLMs to obtain. This makes QAT less feasible compared to other quantization methods for such large-scale models. To address this issue, authors of LLM-QAT [258] generate data from the LLM itself and propose data-free knowledge distillation. In addition to quantizing weights and activations, authors of LLM-QAT also quantize the KV cache in the 7B, 13B, and 30B LLaMA models to 4 bits. Authors of QLoRA [259] introduce a new data type (4-bit NormalFloat) by estimating the quantile of the input tensor through the empirical cumulative distribution function. Besides, they implement Double Quantization, saving approximately 0.37 bits per parameter, equivalent to about 3 GB for a 65B model. Inspired by parameter efficient fine-tuning (PEFT) methods, authors of Parameter-Efficient and Quantization-aware Adaptation (PEQA) [260] decompose the parameter matrix of each fully connected layer into low-bit integers and quantization scales, and the quantization scale can be fine-tuned for different tasks. Authors combine LoRA and QAT to propose Low-Rank Quantization-Aware Training (LR-QAT) [261]. They place the low-rank weights in the integer domain, and apply gradient checkpointing to avoid aggressive memory spikes. Authors of EfficientQAT [262] propose a two-step method to quantize LLMs: block-wise training of all parameters (Block-AP) and end-to-end training of quantization parameters.

Unlike QAT, PTQ is typically training-free and applied after the model has been trained. PTQ is widely used in practice [263]. Authors of GPTQ [264] propose layer-wise quantization and optimal brain quantization method, reducing the bandwidth down to 3 or 4bits per weight while maintaining accuracy performance. Authors of Activation-aware Weight Quantization (AWQ) [265] find that not all weights in LLMs are equally crucial for performance. They introduce a weight-only quantization method that enhances accuracy without requiring additional training by safeguarding the more "important" weights. Additionally, they propose Tinychat, a framework to map and implement AWQ on edge platforms, achieving about 3× speedup to both VILA-7B and VILA-13B on NVIDIA Jetson Orin. Considering that weights are easier to quantize than activations, authors of SmoothQuant [266] address activation outliers by transferring the quantization challenge from activations to weights. They achieve about 1.56× inference acceleration and halve the memory footprint for OPT-13B and OPT-30B with negligible loss in accuracy. Authors of SpQR (Sparse-Quantized Representation (SpQR)) [267] propose the sparse-matrix multiplication and dense-quantized matrix multiplication method to compress LLMs' weights, achieving about 20%–30% faster for LLM generation compared to normal 16-bit inference. Authors of QoQ (Quattuor-Octo-Quattuor) [268] algorithm which quantizes LLMs with 4-bit weight, 8-bit activation, and 4-bit KV caches. They introduce a two-step progressive group quantization: quantize weights to 8 bits using per-channel FP16 scales, then quantize these 8-bit intermediates to 4 bits. In summary, because the training process of LLMs is complex and needs large computational resources, PTQ is widely developed after the LLMs training. Existing well-developed PTQ methods have low loss reduction even near-lossless [269], bringing great benefits to global researchers. Some implemented edge LLM inference and quantization tools and systems include Tinychat [270], QServe [268], AutoGPTQ [271] and Llama.cpp [272], NVIDIA TensorRT-LLM [273], etc.

Knowledge distillation is another way to compress LLMs. KD in LLMs can be divided into two parts: white-box distillation and black-box distillation. White-box distillation needs to access the entire teacher model parameters [214,274,275], while black-box distillation only needs predictions from the teacher model [148,209,276–280]. In black-box distillation, some studies utilize API/Instructions to distil the small student model [281,282]. Authors [283] propose the chain-of-thought (CoT) prompting to improve the reasoning capabilities of LLMs, i.e., generate a series of intermediate reasoning results step-by-step. Many studies (e.g., Fine-tune-CoT) [284–289] generate many reasoning results by CoT from large teacher models, then finetune the small student model with them. Authors [290] propose to combine in-context learning objectives with LLMs so that distilling in-context knowledge to the student model.

8. Future trends

Trustworthy AI. In recent years, AI technologies have experienced tremendous progress and are increasingly deployed across domains (e.g., automatic drive, healthcare, games [291]). SOTA AI models are developed and publicly available, achieving top performance. With the rapid development of technologies such as automated machine learning (AutoML) [188], which can build deep learning models automatically, including network structure design, hyperparameter tuning, etc. However, when people deploy AI models on practical applications, **how to make AI more reliable** is a big challenge [292]. Many efforts have been made to address it. Before deployment, the verification, testing, adversarial attack and defence of the deep model become necessary [293]. At the model level, neural network testing and debugging (e.g., white-Box testing [294], mutation test [295]) has risen. They used neural coverage metrics and mutation operators to test the response of deep learning models. At the data level, out-of-distribution (OOD) data and adversarial samples with imperceptible perturbations can completely fool deep learning models [296]. Adversarial training methods [297] retrain the deep model using adversarial samples to improve robustness. Causal inference [298,299] methods are proposed to decouple the causal representation of deep neural networks for mitigating OOD problems. On the other hand, LLMs' alignment and trustworthy also received recent attention [300]. Unfortunately, most existing research deploys AI models with little consideration of the above practical problems.

Robust deployment. Robustness research in AI is a challenging spotlight [301,302]. In AIoT edge deployment, robust AI models are especially critical because users find it hard to 'believe' vulnerable models. This paper reviews some related works, including AIoT-based edge devices, lightweight DNNs, and neural network compression. However, most existing research deploys AI models with little consideration of **the robustness evaluation after deployment**. After network pruning, network quantization, and knowledge distillation, AI models on edge devices are rarely evaluated in the robustness field. In Tables 8 7 6, most research makes efforts to compress AI models to achieve similar performance against original models on the same dataset. However, the performance between compressed AI models against original models may be completely different on other similar data, which brings the robustness problem on edge deployment. On similar data researchers are concerned, the performance of compressed AI models is very important in practice. While may not be important on similar data researchers are not concerned. Evaluated indicators (e.g., data coverage [292], Non-I.I.D. index [303,304]) and their increment/loss before and after deployment on the edge devices should be considered.

9. Conclusion

This paper focuses on the hardware and software for AIoT edge applications. This paper reviews about 300 articles, where most deploy DL methods on practical devices for real applications. AI edge devices with AI accelerators are analyzed and compared. Lightweight CNNs and LLMs, many neural network compression methods (network Pruning, network quantization, and knowledge distillation) are introduced. Detailed comparison and discussion among AI edge devices and CNNs compression methods are performed. Future trends are presented, including trustworthy AI, LLMs, and robust deployment.

CRediT authorship contribution statement

Kailai Sun: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Xinwei Wang:** Writing – review & editing, Writing – original draft, Visualization, Validation, Resources, Methodology, Investigation. **Xi Miao:** Writing – review & editing, Validation, Investigation. **Qianchuan Zhao:** Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by National Natural Science Foundation of China under Grant No. 62192751, the 111 International Collaboration Program of China under Grant No. BP2018006, in part by the BNRist Program, China under Grant No. BNR2019TD01009.

Appendix

A.1. Metrics for evaluating the scale of neural networks

There are several metrics widely used to evaluate the scale of neural networks: parameters, computational complexity, memory usage, and memory access cost.

A.1.1. Parameters

Parameters [10] can be intuitively understood as the total number of connections in the network. In the fully connected layer, for example, each output is connected to all inputs, and the relationship between the outputs and inputs in a channel is in Eq. (4):

$$Output_i = \sum_{j=1}^{C_{in}} W_{(i,j)} * Input_j \quad (1 \le i \le C_{out}), \tag{4}$$

where the weight W is the parameter, and there are $C_{in} \times C_{out}$. So, the total parameters are $C_{in} \times C_{out}$ (the bias parameters are omitted for simplification).

As for the convolution layer, if a convolution kernel is $(2k+1) \times (2k+1)$ (even-sized convolution kernels cause feature shifts due to sensory field asymmetry [305]), the output of a point on a feature map is in Eq. (5). The total parameters are $C_{in} \times C_{out} \times (2k+1)^2$ [10].

$$Output_{(C_{out},x,y)} = \sum_{C=1}^{C_{in}} \sum_{-k \le i,j \le k} W_{(C,x+i,y+j)} * Input_{(C,x+i,y+j)}. \tag{5}$$

In contrast, other network layers (s.a., pooling layer, activation layer, BN layer) have no parameters or a few parameters. Many CNNs and their variants' parameters focus on the convolutional layer, while the Transformer and its variant (e.g., ViT) focus almost entirely on the fully connected layer. Therefore, most lightweight work based on CNNs focuses on how to reduce the parameters in the convolutional layer. In contrast, the work based on ViT series focuses on how to reduce the parameters in the fully connected layer.

A.1.2. Computational complexity

The parameters are only related to the network structure, not the input size. The computational complexity of the convolutional layer is related to both. Computational complexity was calculated in most research on lightweight DNNs [55,56,110–113,117,306–312]. It refers to the sum of the number of addition and multiplication operations when a network performs a complete derivation of the input data. The fully connected layer (e.g., (4)), requires $C_{in} \times C_{out}$ multiplications and $C_{in} \times C_{out} - 1$ additions to complete the whole computation. Since the computation of multiplication is more complex than that of addition, researchers generally ignore addition and compute multiplication only. Meanwhile, since the mainstream deep neural networks use the Float32 data type, the computational complexity of the fully connected layer can be represented as $C_{in} \times C_{out}$ FLOPs (floating-point operations).

One convolution operation can obtain the output of a point in a channel, requiring $k^2 \times C_{in}$ multiplications. In order to obtain one convolution layer ($C_{out} \times W_{out} \times H_{out}$ is the size of the output feature map after padding and stride), the computational complexity of this convolutional layer is $k^2 \times C_{in} \times C_{out} \times W_{out} \times H_{out}$ FLOPs. Similarly to the parameters, most computational complexity in CNN focuses on the convolutional layer, while ViT focuses on the fully connected layer.

A.1.3. Memory usage

The parameters are a theoretical metric used to quantify the size of the network, while memory usage is the metric for practical deployment. In general, the memory usage of a deep neural network consists of three components: the model itself, the input, output, and intermediate feature maps of the model, and the gradient of the feedback and other parameters. The calculation of memory usage is simple: if the parameters of a model are 100M, and the data format of all parameters is Float32, each parameter occupies 32 bits (4 bytes), and the memory usage of the model itself is 400 MB. For example, if the input is $224 \times 224 \times 3$, and its batchsize is 16, then the occupied memory size is $16 \times 224 \times 224 \times 3 \times 4/1024^2 \approx 9.18$ MB. And if the intermediate feature map is $56 \times 56 \times 512$, then the occupied memory is $16 \times 56 \times 56 \times 512 \times 4/1024^2 = 98$ MB. Because the feedback gradient corresponds to each weight separately, the memory usage of the feedback gradient can be considered related to the model's parameters. If the SGD optimizer with momentum is used, the network needs to store both the gradient ∂ and the corresponding momentum, and the total memory usage of the gradient is twice that of the model. When using the ADAM optimizer, the total memory usage is three times that of the model. The memory used to run the network will be slightly more than the theoretical calculation.

Generally, training networks require a larger batch size, so memory usage for intermediate features increases dramatically. Hence, the memory of training graphics cards is huge (e.g., RTX3090 has 24 GB memory, A100 has 40 GB or 80 GB memory, and H100 has 80 GB memory). However, AI edge devices do not need to train the network, only to infer, so they do not need to store gradients. The memory of the Jetson Nano is 2/4 GB, and the memory of the Coral Dev Board is only 1 GB. But, the problem lies in that the training graphics card has a separate memory, which is independent of the memory occupied by CPU processes. But the CPU and GPU (or VPN, TPU) in edge devices need to share the same memory, so even if the network input batch size is set to small, the memory of most low-cost edge devices is still insufficient.

A.1.4. Memory access cost

In this section, we will answer the question raised in Appendix A.1.2: Why does neural network inference actually take much longer than $\frac{FLOPs}{FLOPS}$? $FLOPs$ is the computational complexity, while $FLOPS$ is the computing power of the device. Memory access cost is one of the main reasons. In Appendix A.1.3, we have mentioned when the network runs, the system will store the model weights and data in memory, but the computation process does not take place in memory. A rough calculation process is as follows: 1. The computational unit (e.g., Nvidia's stream processor) reads the weights and data from memory and stores them in the small cache. 2. The computational unit calculates the results based on the weights and data. 3. The computational unit writes the results from the small cache back to memory. To get the calculation result, we need two memory read/write operations and one calculation operation. In a computer, the speed of memory reading and writing is significantly slower than the speed of computation. This is why the actual inference speed of deep neural networks is much slower than $\frac{FLOPs}{FLOPS}$. Because the whole computation process requires a huge amount of memory for reading and writing operations.

How to calculate the memory access cost? The general MAC calculation method is shown in [313]. And the convolutional layer is calculated: suppose the input size of a convolutional layer is $C_{in} * W_{in} * H_{in}$, the convolutional kernel size is $k * k$, and the output size is $C_{out} * W_{out} * H_{out}$, the computational unit consumes $C_{in} \times W_{in} \times H_{in}$ memory accesses to read the input data and consumes $k^2 \times C_{in} \times C_{out}$ memory accesses to read the weight data and consumes $C_{out} \times W_{out} \times H_{out}$ memory accesses to write the output back to memory. Then, the memory access cost for a convolutional layer is shown in Eq. (6)

$$MAC_{conv} = C_{in} \times W_{in} \times H_{in} + k^2 \times C_{in} \times C_{out} + C_{out} \times W_{out} \times H_{out} \tag{6}$$

For example, a convolutional layer with input feature map size 224×224, input channel 128, output channel 256, convolutional kernel size 3×3 and output size 112×112, its memory access cost is 6.12M+0.28M+3.06M = 9.46M.

The memory access cost of the fully connected layer is calculated as follows: assuming the input channel is C_{in}, and the output channel is C_{out}; then it consumes C_{in} memory accesses to read the input, $(C_{in} + 1) \times C_{out}$ memory accesses to read the weights, and C_{out} memory accesses to write back the output, then the memory access cost of the fully connected layer is calculated as Eq. (7)

$$MAC_{fc} = C_{in} + (C_{in} + 1) \times C_{out} + C_{out} \tag{7}$$

For example, a fully-connected layer with 1000 input channels and 200 output channels, its memory access cost is 1000+(1000+1)×200+200≈ 0.19 M. As we can see, a convolutional layer can compute more features with fewer weights than a fully-connected layer. If a two-dimensional fully-connected operation replaces the convolutional layer, the memory access cost of reading weights is hard to accept.

Of course, the memory access cost calculation is based on the assumption that reading a single value is considered a single memory access operation. However, in practice, current CPUs and GPUs have many optimization methods for reading memory and can sometimes read multiple floating-point numbers in one memory access. The memory access cost can only be used as a theoretical reference, but it also represents the rough scale of a deep neural network.

A.2. 8-Bit quantization process

Take the simple linear quantization of int16 to uint8 as an example. The process is in Eq. (8):

$$\begin{aligned}
&uint8 = round(\frac{int16}{S} + Z),\\
&S = \frac{int16_{max} - int16_{min}}{uint8_{max} - uint8_{min}} = \frac{65535}{255} = 257,\\
&Z = round(uint8_{max} - \frac{int16_{max}}{S}) = 128,\\
&uint8 = round(\frac{int16}{257} + 128),
\end{aligned} \tag{8}$$

- 1. For a standard convolutional layer, first, quantize the input and network weights to Int8, then record the scaling parameters

Table 11
Metrics of different efficient CNN architectures.

CNN architectures	Parameters	Computational complexity	Memory usage	Memory access cost
A standard convolution layer	$k^2 \times C_{in} \times C_{out}$	$k^2 \times C_{in} \times C_{out} \times W_{out} \times H_{out}$	$k^2 \times C_{in} \times C_{out} + W_{out} \times H_{out} \times C_{out}$	$W_{in} \times H_{in} \times C_{in} + k^2 \times C_{in} \times C_{out} + W_{out} \times H_{out} \times C_{out}$
An Inception convolution layer	$C_{in} \times C_{mid} + k^2 \times C_{mid} \times C_{out}$	$W \times H \times (C_{in} \times C_{mid} + k^2 \times C_{mid} \times C_{out})$	$C_{in} \times C_{mid} + k^2 \times C_{mid} \times C_{out} + W \times H \times (C_{mid} + C_{out})$	$C_{in} \times C_{mid} + k^2 \times C_{mid} \times C_{out} + W \times H \times (2 \times C_{mid} + C_{out})$
A depthwise separable convolution layer	$k^2 \times C_{in} + C_{in} \times C_{C_out}$	$W \times H \times (k^2 \times C_{in} + C_{in} \times C_{out})$	$k^2 \times C_{in} + C_{in} \times C_{out} + W \times H \times (C_{in} + C_{out})$	$2 \times W \times H \times C_{in} + k^2 \times C_{in} + W \times H \times C_{out} + C_{in} \times C_{out}$
A group convolution layer	$(k^2 \times C_{in} \times C_{out})/G$	$k^2 \times W_{in} \times H_{in} \times C_{in} \times C_{out}/G$	$k^2 \times C_{in} \times C_{out}/G + W_{out} \times H_{out} \times C_{out}$	$W_{in} \times H_{in} \times C_{in} + k^2 \times C_{in} \times C_{out}/G + W_{out} \times H_{out} \times C_{out}$

Table 12
Examples of efficient CNN architectures.

Ref	Published time	Lightweight network	Structure/Improvement	Parameters	Computational complexity
[113]	2018	ConDenseNet	Learned Group Convolution	0.52M	65M
[308]	2018	ShuffleNet	Group Convolution, Channel Shuffle	1.7M	140M
[309]	2018	ShuffleNetV2	MAC Optimization, Parallelism Improvement.	2.3M	146M
[306]	2018	MobileNetV2	Inverted Residual, Linear Bottleneck	3.4M	300M
[310]	2018	ESPNet	Spatial Pyramid of Dilated Convolution, Point-wise Convolutions	0.33M	
[307]	2019	MobileNetV3	SE Module, H-swish Activation Function	2.5M	56M
[311]	2019	ESPNetV2	Group Point-wise Convolution, Depth-wise Dilated Separable Convolution	1.24M	28M
[312]	2020	GhostNet	Ghost Module	5.2M	141M
[314]	2020	CSPNet	integrating feature maps from the beginning and the end of a network stage	2.73M	190.5M
[315]	2020	MCUNet	Searched by TinyNAS	1.2M	168M
[117]	2021	LeViT	Attention Bias, Hardswish Activation Function, Shrinking Attention Block	7.8M	305M
[316]	2022	SLaK	Large saprese convolution kernel and kernel factorizing	30M	5.0G
[317]	2023	RepViT	modify CNN architecture based on ViT	6.8M	1.1G

S_{input}, $S_{weights}$ and zero points Z_{input}, $Z_{weights}$. The data type of the scaling factor is Float32, and the zero point is Int8.

$$\begin{aligned} S_{input} &= \frac{input_{max} - input_{min}}{uint8_{max} - uint8_{min}}, \\ S_{weights} &= \frac{weights_{max} - weights_{min}}{uint8_{max} - uint8_{min}}, \\ Z_{input} &= round(255 - \frac{input_{max}}{S_{input}}), \\ Z_{Weights} &= round(255 - \frac{weights_{max}}{S_{weights}}). \end{aligned} \tag{9}$$

- 2. Next, use Int8 to simulate the convolution operation under Float32. The convolution operation can be simplified as Eq. (10) (the bias of the convolution operation is omitted for simplifying):

$$output_{float} = weights_{float} * input_{float}. \tag{10}$$

Now we substitute Eq. (8) into Eq. (10) to get:

$$\begin{aligned} &S_{output}(output_{uint} - Z_{output}) = S_{weight} S_{input} * \\ &* (weights_{uint} - Z_{weights})(input_{uint} - Z_{input}), \\ &output_{uint} = \frac{S_{weight} S_{input}}{S_{output}} * \\ &(weights_{uint} - Z_{weights})(input_{uint} - Z_{input}) + Z_{output}. \end{aligned} \tag{11}$$

Data availability

No data was used for the research described in the article.

References

[1] M. Shirer, C. MacGillivray, The growth in connected IoT devices is expected to generate 79.4 zb of data in 2025, according to a new IDC forecast, 2019, Available: https://www.iotcentral.io/blog/iot-is-not-a-buzzword-but-necessity.

[2] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Netw. 61 (2015) 85–117, URL https://www.sciencedirect.com/science/article/pii/S0893608014002135.

[3] Z. Chang, S. Liu, X. Xiong, Z. Cai, G. Tu, A survey of recent advances in edge-computing-powered artificial intelligence of things, IEEE Internet Things J. 8 (18) (2021).

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, 2014, arXiv preprint arXiv:1406.1078.

[5] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90.

[6] H. Li, Z. Lin, X. Shen, J. Brandt, G. Hua, A convolutional neural network cascade for face detection, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5325–5334.

[7] S. Dong, P. Wang, K. Abbas, A survey on deep learning and its applications, Comp. Sci. Rev. 40 (2021) 100379.

[8] A. Liu, P. Luh, K. Sun, Integrating machine learning and mathematical optimization for job shop scheduling, 2022.

[9] G. Halhoul Merabet, M. Essaaidi, M. Ben Haddou, B. Qolomany, J. Qadir, M. Anan, A. Al-Fuqaha, M.R. Abid, D. Benhaddou, Intelligent building control systems for thermal comfort and energy-efficiency: A systematic review of artificial intelligence-assisted techniques, Renew. Sustain. Energy Rev. 144 (2021) 110969.

[10] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, L. Jackel, Handwritten digit recognition with a back-propagation network, Adv. Neural Inf. Process. Syst. 2 (1989).

[11] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2015, CoRR, arXiv:1512.03385, URL http://arxiv.org/abs/1512.03385.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, 2017, CoRR, arXiv:1706.03762, URL http://arxiv.org/abs/1706.03762.

[13] S. Zhu, K. Ota, M. Dong, Energy-efficient artificial intelligence of things with intelligent edge, IEEE Internet Things J. 9 (10) (2022) 7525–7532.

[14] C. Zhang, X. Yuan, Q. Zhang, G. Zhu, L. Cheng, N. Zhang, Toward tailored models on private AIoT devices: Federated direct neural architecture search, IEEE Internet Things J. 9 (18) (2022) 17309–17322.

[15] J. Zhang, D. Tao, Empowering things with intelligence: A survey of the progress, challenges, and opportunities in artificial intelligence of things, IEEE Internet Things J. 8 (10) (2021) 7789–7817.

[16] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: A survey, IEEE Internet Things J. 5 (1) (2018) 450–465.

[17] G.C. Marinó, A. Petrini, D. Malchiodi, M. Frasca, Deep neural networks compression: A comparative survey and choice recommendations, Neurocomputing 520 (2023) 152–170.
[18] L. Deng, G. Li, S. Han, L. Shi, Y. Xie, Model compression and hardware acceleration for neural networks: A comprehensive survey, Proc. IEEE 108 (4) (2020) 485–532.
[19] R. Mishra, H. Gupta, Transforming large-size to lightweight deep neural networks for IoT applications, ACM Comput. Surv. 55 (11) (2023).
[20] G. Armeniakos, G. Zervakis, D. Soudris, J. Henkel, Hardware approximate techniques for deep neural network accelerators: A survey, ACM Comput. Surv. 55 (4) (2022).
[21] K.W. Martin, Digital Integrated Circuit Design, Oxford University Press, New York, 2000.
[22] I.J.O. Nano, Nvidia, 2022, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/.
[23] K.P. Seng, P.J. Lee, L.M. Ang, Embedded intelligence on FPGA: Survey, applications and challenges, Electronics 10 (8) (2021) 895.
[24] J.-Y. Li, W.-C. Fang, An edge AI accelerator design based on HDC model for real-time EEG-based emotion recognition system with RISC-V FPGA platform, in: 2024 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2024, pp. 1–5.
[25] T.-K. Chi, T.-Y. Chen, Y.-C. Lin, T.-L. Lin, J.-T. Zhang, C.-L. Lu, S.-L. Chen, K.-C. Li, P.A.R. Abu, An edge computing system with AMD Xilinx FPGA AI customer platform for advanced driver assistance system, Sensors 24 (10) (2024) 3098.
[26] D.L.T. Wong, Y. Li, D. John, W.K. Ho, C.H. Heng, Resource and energy efficient implementation of ECG classifier using binarized CNN for edge AI devices, in: 2021 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2021, pp. 1–5.
[27] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, D. Chen, FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge, in: Proceedings of the 56th Annual Design Automation Conference 2019, 2019, pp. 1–6.
[28] S. Gu, X. Chen, W. Zeng, X. Wang, A deep learning tennis ball collection robot and the implementation on nvidia jetson tx1 board, in: 2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM, IEEE, 2018, pp. 170–175.
[29] B. Ramalingam, A.A. Hayat, M.R. Elara, B. Félix Gómez, L. Yi, T. Pathmakumar, M.M. Rayguru, S. Subramanian, Deep learning based pavement inspection using self-reconfigurable robot, Sensors 21 (8) (2021) 2595.
[30] Z. Wang, W. Ren, Q. Qiu, Lanenet: Real-time lane detection networks for autonomous driving, 2018, arXiv preprint arXiv:1807.01726.
[31] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, V. Sze, Fastdepth: Fast monocular depth estimation on embedded systems, in: 2019 International Conference on Robotics and Automation, ICRA, IEEE, 2019, pp. 6101–6108.
[32] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, H. Adam, NetAdapt: Platform-aware neural network adaptation for mobile applications, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 285–300.
[33] L. Fang, Y. Wu, Y. Li, H. Guo, H. Zhang, X. Wang, R. Xi, J. Hou, Using channel and network layer pruning based on deep learning for real-time detection of ginger images, Agriculture 11 (12) (2021) 1190.
[34] Y. Shao, X. Zhang, H. Chu, X. Zhang, D. Zhang, Y. Rao, AIR-YOLOv3: Aerial infrared pedestrian detection via an improved YOLOv3 with network pruning, Appl. Sci. 12 (7) (2022) 3627.
[35] Y. Tu, Y. Lin, Deep neural network compression technique towards efficient digital signal modulation recognition in edge device, IEEE Access 7 (2019) 58113–58119.
[36] S. Gong, H. Zhou, F. Xue, C. Fang, Y. Li, Y. Zhou, FastRoadSeg: Fast monocular road segmentation network, IEEE Trans. Intell. Transp. Syst. (2022).
[37] X. Xu, S. Caulfield, J. Amaro, G. Falcao, D. Moloney, 1.2 Watt classification of 3D voxel based point-clouds using a CNN on a neural compute stick, Neurocomputing 393 (2020) 165–174.
[38] S. Liu, L.T. Yang, X. Tu, R. Li, C. Xu, Lightweight monocular depth estimation on edge devices, IEEE Internet Things J. 9 (17) (2022) 16168–16180.
[39] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International Conference on Machine Learning, PMLR, 2018, pp. 1587–1596.
[40] T. Ringwald, L. Sommer, A. Schumann, J. Beyerer, R. Stiefelhagen, UAV-Net: A fast aerial vehicle detector for mobile platforms, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2019.
[41] Q. Zheng, X. Tian, M. Yang, H. Su, CLMIP: Cross-layer manifold invariance based pruning method of deep convolutional neural network for real-time road type recognition, Multidimens. Syst. Signal Process. 32 (1) (2021) 239–262.
[42] H. Wu, Y. Hua, H. Zou, G. Ke, A lightweight network for vehicle detection based on embedded system, J. Supercomput. (2022) 1–16.
[43] H. Pan, D. Badawi, A.E. Cetin, Computationally efficient wildfire detection method using a deep convolutional network pruned via fourier analysis, Sensors 20 (10) (2020) 2891.
[44] S. Ullah, D.-H. Kim, Lightweight driver behavior identification model with sparse learning on in-vehicle can-bus sensor data, Sensors 20 (18) (2020) 5030.
[45] Z. Xu, J. Li, Y. Meng, X. Zhang, CAP-YOLO: Channel attention based pruning YOLO for coal mine real-time intelligent monitoring, Sensors 22 (12) (2022) 4331.
[46] Y. Gong, Z. Zhan, Z. Li, W. Niu, X. Ma, W. Wang, B. Ren, C. Ding, X. Lin, X. Xu, et al., A privacy-preserving-oriented DNN pruning and mobile acceleration framework, in: Proceedings of the 2020 on Great Lakes Symposium on VLSI, 2020, pp. 119–124.
[47] S. Zhou, M. Xie, Y. Jin, F. Miao, C. Ding, An end-to-end multi-task object detection using embedded gpu in autonomous driving, in: 2021 22nd International Symposium on Quality Electronic Design, ISQED, IEEE, 2021, pp. 122–128.
[48] C. Yang, P. Zhao, Y. Li, W. Niu, J. Guan, H. Tang, M. Qin, B. Ren, X. Lin, Y. Wang, Pruning parameterization with bi-level optimization for efficient semantic segmentation on the edge, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 15402–15412.
[49] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain., Psychol. Rev. 65 (6) (1958) 386.
[50] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Systems 2 (4) (1989) 303–314.
[51] F. Rosenblatt, Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms, Tech. Rep., Cornell Aeronautical Lab Inc Buffalo NY, 1961.
[52] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning Internal Representations by Error Propagation, Tech. Rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
[53] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[54] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[55] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
[56] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.
[57] P. Sinha, J.W. Gichoya, S. Purkayastha, Leapfrogging medical AI in low-resource contexts using edge tensor processing unit, in: 2022 IEEE Healthcare Innovations and Point of Care Technologies, HI-POCT, IEEE, 2022, pp. 67–70.
[58] P. Sertic, A. Alahmar, T. Akilan, M. Javorac, Y. Gupta, Intelligent real-time face-mask detection system with hardware acceleration for COVID-19 mitigation, in: Healthcare, vol. 10, (5) 2022, p. 873.
[59] A.S.P. de Aguiar, F.B.N. dos Santos, L.C.F. dos Santos, V.M. de Jesus Filipe, A.J.M. de Sousa, Vineyard trunk detection using deep learning—An experimental device benchmark, Comput. Electron. Agric. 175 (2020) 105535.
[60] K. Blekos, S. Nousias, A.S. Lalos, Efficient automated U-net based tree crown delineation using UAV multi-spectral imagery on embedded devices, in: 2020 IEEE 18th International Conference on Industrial Informatics, INDIN, 1, IEEE, 2020, pp. 541–546.
[61] X. Yue, H. Li, M. Shimizu, S. Kawamura, L. Meng, YOLO-GD: A deep learning-based object detection algorithm for empty-dish recycling robots, Machines 10 (5) (2022) 294.
[62] H.-H. Nguyen, D.N.-N. Tran, J.W. Jeon, Towards real-time vehicle detection on edge devices with nvidia jetson tx2, in: 2020 IEEE International Conference on Consumer Electronics-Asia, ICCE-Asia, IEEE, 2020, pp. 1–4.
[63] D. Feng, Research and Implementation of Deep Learning Based Defect Detection Algorithm for Industrial CT Workpiece Perspective View (Master's thesis), 2022.
[64] Y. Cheng, G. Li, N. Wong, H.-B. Chen, H. Yu, DEEPEYE: A deeply tensor-compressed neural network hardware accelerator, in: 2019 IEEE/ACM International Conference on Computer-Aided Design, ICCAD, IEEE, 2019, pp. 1–8.
[65] C. Li, R. Xu, Y. Lv, Y. Zhao, W. Jing, Edge real-time object detection and DPU-based hardware implementation for optical remote sensing images, Remote Sens. 15 (16) (2023) 3975.
[66] M. Ayazoglu, Extremely lightweight quantization robust real-time single-image super resolution for mobile devices, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 2472–2479.
[67] A.S. Aguiar, S.A. Magalhães, F.N. Dos Santos, L. Castro, T. Pinho, J. Valente, R. Martins, J. Boaventura-Cunha, Grape bunch detection at different growth stages using deep learning quantized models, Agronomy 11 (9) (2021) 1890.
[68] T. van Rozendaal, T. Singhal, H. Le, G. Sautiere, A. Said, K. Buska, A. Raha, D. Kalatzis, H. Mehta, F. Mayer, et al., MobileNVC: Real-time 1080p neural video compression on a mobile device, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024, pp. 4323–4333.
[69] S. Ashfaq, M. AskariHemmat, S. Sah, E. Saboori, O. Mastropietro, A. Hoffman, Accelerating deep learning model inference on arm cpus with ultra-low bit quantization and runtime, 2022, arXiv preprint arXiv:2207.08820.
[70] S.-E. Chang, Y. Li, M. Sun, R. Shi, H.K.-H. So, X. Qian, Y. Wang, X. Lin, Mix and match: A novel FPGA-centric deep neural network quantization framework, in: HPCA, IEEE, 2021, pp. 208–220.

[71] Y. Wang, X. Li, M. Shi, K. Xian, Z. Cao, Knowledge distillation for fast and accurate monocular depth estimation on mobile devices, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 2457–2465.

[72] F. Aleotti, M. Poggi, F. Tosi, S. Mattoccia, Learning end-to-end scene flow by distilling single tasks knowledge, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, (07) 2020, pp. 10435–10442.

[73] H. Ahuja, S. Saurav, S. Srivastava, C. Shekhar, Driver drowsiness detection using knowledge distillation technique for real time scenarios, in: 2020 IEEE 17th India Council International Conference, INDICON, IEEE, 2020, pp. 1–5.

[74] P.-E. Sarlin, F. Debraine, M. Dymczyk, R. Siegwart, C. Cadena, Leveraging deep visual descriptors for hierarchical efficient localization, in: Conference on Robot Learning, PMLR, 2018, pp. 456–465.

[75] F. Aleotti, G. Zaccaroni, L. Bartolomei, M. Poggi, F. Tosi, S. Mattoccia, Real-time single image depth perception in the wild with handheld devices, Sensors 21 (1) (2020) 15.

[76] R. Jin, Q. Niu, Automatic fabric defect detection based on an improved YOLOv5, Math. Probl. Eng. (2021).

[77] K. Su, C.M.D. Intisar, Q. Zhao, Y. Tomioka, Knowledge distillation for real-time on-road risk detection, in: 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, IEEE, 2020, pp. 110–117.

[78] Y.-C. Yoon, H. Jung, Real-time multi-person action recognition with a neural compute stick, in: 2021 21st International Conference on Control, Automation and Systems, ICCAS, IEEE, 2021, pp. 1135–1140.

[79] F. Guzzi, L. De Bortoli, R.S. Molina, S. Marsi, S. Carrato, G. Ramponi, Distillation of an end-to-end oracle for face verification and recognition sensors, Sensors 20 (5) (2020) 1369.

[80] J. Wu, Research and Implementation of Low Illumination Image Enhancement Model Compression Algorithm for Embedded Devices (Ph.D. thesis), 2021.

[81] A. Chen, Research on Edge Computing Technology Oriented to Object Detection Execution Efficiency Optimization (Ph.D. thesis), 2021.

[82] Q. Zhao, Research and Implementation on the Deep Learning Object Detection Model Accelerator Based on Zynq-7000 (Ph.D. thesis), 2021.

[83] Z. Lingling, C. Fucai, G. Chao, Improvement of face detection algorithm based on lightweight convolutional neural network, in: 2020 IEEE 6th International Conference on Computer and Communications, ICCC, 2020, pp. 1191–1197.

[84] L. Zhu, Design and Implementation of Real-Time Face Detection Algorithm for Mobile Platform (Ph.D. thesis), 2021.

[85] M. Sepahvand, F. Abdali-Mohammadi, A. Taherkordi, An adaptive teacher–student learning algorithm with decomposed knowledge distillation for on-edge intelligence, Eng. Appl. Artif. Intell. 117 (2023) 105560.

[86] M. Bharadhwaj, G. Ramadurai, B. Ravindran, Detecting vehicles on the edge: Knowledge distillation to improve performance in heterogeneous road traffic, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 3192–3198.

[87] N. Aghli, E. Ribeiro, Combining weight pruning and knowledge distillation for cnn compression, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 3191–3198.

[88] J.-C. Zheng, S.-D. Sun, S.-J. Zhao, Fast ship detection based on lightweight YOLOv5 network, IET Image Process. 16 (6) (2022) 1585–1593.

[89] J. Vandendriessche, N. Wouters, B. da Silva, M. Lamrini, M.Y. Chkouri, A. Touhafi, Environmental sound recognition on embedded systems: From FPGAs to TPUs, Electronics 10 (21) (2021) 2622.

[90] M. Rizk, D. Heller, R. Douguet, A. Baghdadi, J.-P. Diguet, Optimization of deep-learning detection of humans in marine environment on edge devices, in: ICECS 2022: IEEE International Conference on Electronics Circuits and Systems, 2022.

[91] E. Lygouras, N. Santavas, A. Taitzoglou, K. Tarchanidis, A. Mitropoulos, A. Gasteratos, Unsupervised human detection with an embedded vision system on a fully autonomous UAV for search and rescue operations, Sensors 19 (16) (2019) 3542.

[92] B. Sudharsan, S. Malik, P. Corcoran, P. Patel, J.G. Breslin, M.I. Ali, OWSNet: Towards real-time offensive words spotting network for consumer iot devices, in: 2021 IEEE 7th World Forum on Internet of Things, WF-IoT, IEEE, 2021, pp. 83–88.

[93] B. Wang, X. Peng, M. Jiang, D. Liu, Real-time fault detection for UAV based on model acceleration engine, IEEE Trans. Instrum. Meas. 69 (12) (2020) 9505–9516.

[94] D.A. Ron, P.J. Freire, J.E. Prilepsky, M. Kamalian-Kopae, A. Napoli, S.K. Turitsyn, Experimental implementation of a neural network optical channel equalizer in restricted hardware using pruning and quantization, Sci. Rep. 12 (1) (2022) 1–14.

[95] S. Wang, J. Zhao, N. Ta, X. Zhao, M. Xiao, H. Wei, A real-time deep learning forest fire monitoring algorithm based on an improved pruned+ KD model, J. Real-Time Image Process. 18 (6) (2021) 2319–2329.

[96] J. Dong, K. Ota, M. Dong, Real-time survivor detection in UAV thermal imagery based on deep learning, in: 2020 16th International Conference on Mobility, Sensing and Networking, MSN, IEEE, 2020, pp. 352–359.

[97] Z. Dou, D. Ye, B. Wang, AutoSegEdge: Searching for the edge device real-time semantic segmentation based on multi-task learning, Image Vis. Comput. (2023) 104719.

[98] C. Yu, T. Chen, Z. Gan, J. Fan, Boost vision transformer with GPU-friendly sparsity and quantization, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 22658–22668.

[99] L. Fu, K. Yan, Y. Zhang, R. Chen, Z. Ma, F. Xu, T. Zhu, EdgeCog: A real-time bearing fault diagnosis system based on lightweight edge computing, IEEE Trans. Instrum. Meas. (2023).

[100] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, S. Han, GAN compression: Efficient architectures for interactive conditional gans, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 5284–5294.

[101] S. Angarano, F. Salvetti, M. Martini, M. Chiaberge, Generative adversarial super-resolution at the edge with knowledge distillation, Eng. Appl. Artif. Intell. 123 (2023) 106407.

[102] B. Yao, L. Liu, Y. Peng, X. Peng, Intelligent measurement on edge devices using hardware memory-aware joint compression enabled neural networks, IEEE Trans. Instrum. Meas. (2023).

[103] G. Jose, A. Kumar, S. Kruthiventi SS, S. Saha, H. Muralidhara, Real-time object detection on low power embedded platforms, in: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, 2019.

[104] L.-Y. Liew, S.-D. Wang, Object detection edge performance optimization on FPGA-based heterogeneous multiprocessor systems, in: 2022 IEEE International Conference on Consumer Electronics, ICCE, IEEE, 2022, pp. 1–6.

[105] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., An image is worth 16 × 16 words: Transformers for image recognition at scale, 2020, arXiv preprint arXiv:2010.11929.

[106] S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, A survey of deep learning techniques for autonomous driving, J. Field Robotics 37 (3) (2020) 362–386.

[107] T. Czimmermann, G. Ciuti, M. Milazzo, M. Chiurazzi, S. Roccella, C.M. Oddo, P. Dario, Visual-based defect detection and classification approaches for industrial applications—A survey, Sensors 20 (5) (2020) 1459.

[108] Y. Tang, M. Chen, C. Wang, L. Luo, J. Li, G. Lian, X. Zou, Recognition and localization methods for vision-based fruit picking robots: A review, Front. Plant Sci. 11 (2020) 510.

[109] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Adv. Neural Inf. Process. Syst. 33 (2020) 1877–1901.

[110] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size, 2016, arXiv preprint arXiv:1602.07360.

[111] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

[112] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017, arXiv preprint arXiv:1704.04861.

[113] G. Huang, S. Liu, L. Van der Maaten, K.Q. Weinberger, ConDenseNet: An efficient densenet using learned group convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2752–2761.

[114] X. Liu, J. Zhao, J. Li, B. Cao, Z. Lv, Federated neural architecture search for medical data security, IEEE Trans. Ind. Inf. 18 (8) (2022) 5628–5636.

[115] A. Vaswani, P. Ramachandran, A. Srinivas, N. Parmar, B. Hechtman, J. Shlens, Scaling local self-attention for parameter efficient visual backbones, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 12894–12904.

[116] Y. Chen, X. Dai, D. Chen, M. Liu, X. Dong, L. Yuan, Z. Liu, Mobile-former: Bridging mobilenet and transformer, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 5270–5279.

[117] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, M. Douze, LeViT: A vision transformer in convnet's clothing for faster inference, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 12259–12269.

[118] M.C. Mozer, P. Smolensky, Skeletonization: A technique for trimming the fat from a network via relevance assessment, Adv. Neural Inf. Process. Syst. 1 (1988).

[119] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, Adv. Neural Inf. Process. Syst. 2 (1989).

[120] B. Hassibi, D.G. Stork, G.J. Wolff, Optimal brain surgeon and general network pruning, in: IEEE International Conference on Neural Networks, IEEE, 1993, pp. 293–299.

[121] T. Liang, J. Glossner, L. Wang, S. Shi, X. Zhang, Pruning and quantization for deep neural network acceleration: A survey, Neurocomputing 461 (2021) 370–403.

[122] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, W.J. Dally, EIE: Efficient inference engine on compressed deep neural network, ACM SIGARCH Comput. Archit. News 44 (3) (2016) 243–254.

[123] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S.W. Keckler, W.J. Dally, SCNN: An accelerator for compressed-sparse convolutional neural networks, ACM SIGARCH Comput. Archit. News 45 (2) (2017) 27–40.

[124] Q. Qin, J. Ren, J. Yu, H. Wang, L. Gao, J. Zheng, Y. Feng, J. Fang, Z. Wang, To compress, or not to compress: Characterizing deep learning model compression for embedded inference, in: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing, IEEE, 2018, pp. 729–736.

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

[126] M. Zhu, Y. Tang, K. Han, Vision transformer pruning, 2021, arXiv preprint arXiv:2104.08500.

[127] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient convnets, 2016, arXiv preprint arXiv:1608.08710.

[128] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4340–4349.

[129] H. Hu, R. Peng, Y.-W. Tai, C.-K. Tang, Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, 2016, arXiv preprint arXiv:1607.03250.

[130] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2736–2744.

[131] J.-H. Luo, J. Wu, An entropy-based pruning method for cnn compression, 2017, arXiv preprint arXiv:1706.05791.

[132] L. Theis, I. Korshunova, A. Tejani, F. Huszár, Faster gaze prediction with dense networks and fisher pruning, 2018, arXiv preprint arXiv:1801.05787.

[133] N. Lee, T. Ajanthan, P.H. Torr, SNIP: Single-shot network pruning based on connection sensitivity, 2018, arXiv preprint arXiv:1810.02340.

[134] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, L. Shao, Hrank: Filter pruning using high-rank feature map, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1529–1538.

[135] J.-H. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5058–5066.

[136] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.

[137] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V.I. Morariu, X. Han, M. Gao, C.-Y. Lin, L.S. Davis, NISP: Pruning networks using neuron importance score propagation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9194–9203.

[138] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, S. Han, AMC: Automl for model compression and acceleration on mobile devices, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 784–800.

[139] J. Yu, T. Huang, Autoslim: Towards one-shot architecture search for channel numbers, 2019, arXiv preprint arXiv:1903.11728.

[140] G. Li, X. Ma, X. Wang, H. Yue, J. Li, L. Liu, X. Feng, J. Xue, Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning, J. Syst. Archit. 124 (2022) 102431.

[141] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks, Adv. Neural Inf. Process. Syst. 29 (2016).

[142] Z. Huang, N. Wang, Data-driven sparse structure selection for deep neural networks, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 304–320.

[143] G. Li, X. Ma, X. Wang, L. Liu, J. Xue, X. Feng, Fusion-catalyzed pruning for optimizing deep learning on intelligent edge devices, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11) (2020) 3614–3626.

[144] Z. Tanluren, yolov3-channel-and-layer-pruning, 2019, https://github.com/tanluren/yolov3-channel-and-layer-pruning.

[145] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, Adv. Neural Inf. Process. Syst. 28 (2015).

[146] T. Gale, E. Elsen, S. Hooker, The state of sparsity in deep neural networks, 2019, arXiv preprint arXiv:1902.09574.

[147] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2018, arXiv preprint arXiv:1803.03635.

[148] C. Wang, G. Zhang, R. Grosse, Picking winning tickets before training by preserving gradient flow, 2020, arXiv preprint arXiv:2002.07376.

[149] H. Tanaka, D. Kunin, D.L. Yamins, S. Ganguli, Pruning neural networks without any data by iteratively conserving synaptic flow, Adv. Neural Inf. Process. Syst. 33 (2020) 6377–6389.

[150] J. Frankle, G.K. Dziugaite, D.M. Roy, M. Carbin, Pruning neural networks at initialization: Why are we missing the mark? 2020, arXiv preprint arXiv:2009.08576.

[151] C. Gamanayake, L. Jayasinghe, B.K.K. Ng, C. Yuen, Cluster pruning: An efficient filter pruning method for edge ai vision applications, IEEE J. Sel. Top. Sign. Proces. 14 (4) (2020) 802–816.

[152] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, Y. Yang, Asymptotic soft filter pruning for deep convolutional neural networks, IEEE Trans. Cybern. 50 (8) (2019) 3594–3604.

[153] W. Kahan, IEEE standard 754 for binary floating-point arithmetic, in: Lecture Notes on the Status of IEEE, vol. 754, (94720–1776) 1996, p. 11.

[154] V. Vanhoucke, A. Senior, M.Z. Mao, Improving the speed of neural networks on CPUs, 2011.

[155] T. Dettmers, 8-bit approximations for parallelism in deep learning, 2015, arXiv preprint arXiv:1511.04561.

[156] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: International Conference on Machine Learning, PMLR, 2015, pp. 1737–1746.

[157] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2704–2713.

[158] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: International Conference on Machine Learning, PMLR, 2015, pp. 1737–1746.

[159] D. Das, N. Mellempudi, D. Mudigere, D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, et al., Mixed precision training of convolutional neural networks using integer operations, 2018, arXiv preprint arXiv:1802.00930.

[160] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, et al., Mixed precision training, 2017, arXiv preprint arXiv:1710.03740.

[161] R. Banner, Y. Nahshan, D. Soudry, Post training 4-bit quantization of convolutional networks for rapid-deployment, Adv. Neural Inf. Process. Syst. 32 (2019).

[162] A. Mishra, E. Nurvitadhi, J.J. Cook, D. Marr, WRPN: Wide reduced-precision networks, 2017, arXiv preprint arXiv:1709.01134.

[163] M. Vandersteegen, K. Van Beeck, T. Goedemé, Integer-only CNNs with 4 bit weights and bit-shift quantization scales at full-precision accuracy, Electronics 10 (22) (2021) 2823.

[164] C. Leng, Z. Dou, H. Li, S. Zhu, R. Jin, Extremely low bit neural network: Squeeze the last bit out with admm, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[165] J. Choi, S. Venkataramani, V.V. Srinivasan, K. Gopalakrishnan, Z. Wang, P. Chuang, Accurate and efficient 2-bit quantized neural networks, Proc. Mach. Learn. Syst. 1 (2019) 348–359.

[166] J. Choi, P.I.-J. Chuang, Z. Wang, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Bridging the accuracy gap for 2-bit quantized neural networks (QNN), 2018, arXiv preprint arXiv:1807.06964.

[167] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1, 2016, arXiv preprint arXiv:1602.02830.

[168] H. Phan, Y. He, M. Savvides, Z. Shen, et al., Mobinet: A mobile binary network for image classification, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2020, pp. 3453–3462.

[169] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, J. Song, Forward and backward information retention for accurate binary neural networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 2250–2259.

[170] F. Li, B. Zhang, B. Liu, Ternary weight networks, 2016, arXiv preprint arXiv:1605.04711.

[171] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu, et al., Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml, Mach. Learn.: Sci. Technol. 2 (1) (2020) 015001.

[172] S.A. Tailor, J. Fernandez-Marques, N.D. Lane, Degree-quant: Quantization-aware training for graph neural networks, 2020, arXiv preprint arXiv:2008.05000.

[173] K. Wang, Z. Liu, Y. Lin, J. Lin, S. Han, HAQ: Hardware-aware automated quantization with mixed precision, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 8612–8620.

[174] G. Hinton, O. Vinyals, J. Dean, et al., Distilling the knowledge in a neural network, 2, (7) 2015, arXiv preprint arXiv:1503.02531.

[175] J. Gou, B. Yu, S.J. Maybank, D. Tao, Knowledge distillation: A survey, Int. J. Comput. Vis. 129 (6) (2021) 1789–1819.

[176] L. Wang, K.-J. Yoon, Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks, IEEE Trans. Pattern Anal. Mach. Intell. (2021).

[177] G. Chen, W. Choi, X. Yu, T. Han, M. Chandraker, Learning efficient object detection models with knowledge distillation, Adv. Neural Inf. Process. Syst. 30 (2017).

[178] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: Hints for thin deep nets, 2014, arXiv preprint arXiv:1412.6550.

[179] F. Tung, G. Mori, Similarity-preserving knowledge distillation, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1365–1374.

[180] N. Passalis, A. Tefas, Learning deep representations with probabilistic knowledge transfer, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 268–284.

[181] S. Ahn, S.X. Hu, A. Damianou, N.D. Lawrence, Z. Dai, Variational information distillation for knowledge transfer, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 9163–9171.

[182] B. Heo, M. Lee, S. Yun, J.Y. Choi, Knowledge transfer via distillation of activation boundaries formed by hidden neurons, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, (01) 2019, pp. 3779–3787.

[183] A. Tarvainen, H. Valpola, Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results, Adv. Neural Inf. Process. Syst. 30 (2017).

[184] C. Shen, X. Wang, Y. Yin, J. Song, S. Luo, M. Song, Progressive network grafting for few-shot knowledge distillation, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, (3) 2021, pp. 2541–2549.

[185] S. You, C. Xu, C. Xu, D. Tao, Learning from multiple teacher networks, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 1285–1294.

[186] Y. Liu, W. Zhang, J. Wang, Adaptive multi-teacher multi-level knowledge distillation, Neurocomputing 415 (2020) 106–113.

[187] W. Son, J. Na, J. Choi, W. Hwang, Densely guided knowledge distillation using multiple teacher assistants, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 9395–9404.

[188] A. Singh, J. Amutha, J. Nagar, S. Sharma, C.-C. Lee, AutoML-ID: Automated machine learning model for intrusion detection using wireless sensor network, Sci. Rep. 12 (2022).

[189] A. Dequino, F. Conti, L. Benini, ViT-LR: Pushing the envelope for transformer-based on-device embedded continual learning.

[190] I. TensorRT, TensorRT, 2024, https://developer.nvidia.com/blog/accelerate-generative-ai-inference-performance-with-nvidia-tensorrt-model-optimizer-now-publicly-available/.

[191] J.-H. Luo, J. Wu, Neural network pruning with residual-connections and limited-data, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1458–1467.

[192] J. Kim, Y. Bhalgat, J. Lee, C. Patel, N. Kwak, QKD: Quantization-aware knowledge distillation, 2019, arXiv preprint arXiv:1911.12491.

[193] B. Zhuang, L. Liu, M. Tan, C. Shen, I. Reid, Training quantized neural networks with a full-precision auxiliary module, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1488–1497.

[194] R. Miles, K. Mikolajczyk, Cascaded channel pruning using hierarchical self-distillation, 2020, arXiv preprint arXiv:2008.06814.

[195] B. Zhuang, C. Shen, M. Tan, L. Liu, I. Reid, Towards effective low-bitwidth convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7920–7928.

[196] Ultralytics, YOLOv5, 2023, https://github.com/ultralytics/yolov5. (Accessed 8 2023).

[197] H. Naveed, A.U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Barnes, A. Mian, A comprehensive overview of large language models, 2023, arXiv preprint arXiv:2307.06435.

[198] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, H. Jin, T. Chen, Z. Jia, Towards efficient generative large language model serving: A survey from algorithms to systems, 2023, arXiv preprint arXiv:2312.15234.

[199] E. Strubell, A. Ganesh, A. McCallum, Energy and policy considerations for deep learning in NLP, 2019, arXiv preprint arXiv:1906.02243.

[200] Z. Tang, Y. Wang, X. He, L. Zhang, X. Pan, Q. Wang, R. Zeng, K. Zhao, S. Shi, B. He, et al., FusionAI: Decentralized training and deploying LLMs with massive consumer-level GPUs, 2023, arXiv preprint arXiv:2309.01172.

[201] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Adv. Neural Inf. Process. Syst. 33 (2020) 1877–1901.

[202] S. Smith, M. Patwary, Norick, et al., Using deepspeed and megatron to train megatron-turing NLG 530B, a large-scale generative language model, 2022, arXiv preprint arXiv:2201.11990.

[203] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D.d.L. Casas, L.A. Hendricks, J. Welbl, A. Clark, et al., Training compute-optimal large language models, 2022, arXiv preprint arXiv:2203.15556.

[204] A. Zeng, X. Liu, Z. Du, Z. Wang, H. Lai, M. Ding, Z. Yang, Y. Xu, W. Zheng, X. Xia, et al., GLM-130B: An open bilingual pre-trained model, 2022, arXiv preprint arXiv:2210.02414.

[205] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A.M. Dai, A. Hauth, et al., Gemini: A family of highly capable multimodal models, 2023, arXiv preprint arXiv:2312.11805.

[206] B. Workshop, T.L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A.S. Luccioni, F. Yvon, et al., Bloom: A 176b-parameter open-access multilingual language model, 2022, arXiv preprint arXiv:2211.05100.

[207] X. Geng, A. Gudibande, H. Liu, E. Wallace, P. Abbeel, S. Levine, D. Song, Koala: A dialogue model for academic research, 2023, Blog post, April, vol. 1.

[208] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, T.B. Hashimoto, Alpaca: A strong, replicable instruction-following model, Stanf. Cent. Res. Found. Model. 3 (6) (2023) 7, https://crfm.stanford.edu/2023/03/13/alpaca.html.

[209] S. Semnani, V. Yao, H. Zhang, M. Lam, WikiChat: Stopping the hallucination of large language model chatbots by few-shot grounding on Wikipedia, in: Findings of the Association for Computational Linguistics, EMNLP 2023, 2023, pp. 2387–2413.

[210] H. Liu, C. Li, Y. Li, Y.J. Lee, Improved baselines with visual instruction tuning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 26296–26306.

[211] OpenAI, GPT-4o, 2024, https://openai.com/index/hello-gpt-4o/.

[212] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., LLaMA: Open and efficient foundation language models, 2023, arXiv preprint arXiv:2302.13971.

[213] Meta, Llama3.1, 2024, https://llama.meta.com/.

[214] X. Zhu, J. Li, Y. Liu, C. Ma, W. Wang, A survey on model compression for large language models, 2023, arXiv preprint arXiv:2308.07633.

[215] F. Catania, M. Spitale, F. Garzotto, Conversational agents in therapeutic interventions for neurodevelopmental disorders: A survey, ACM Comput. Surv. 55 (10) (2023) 1–34.

[216] S. Goyal, A.R. Choudhury, S. Raje, V. Chakaravarthy, Y. Sabharwal, A. Verma, PoWER-BERT: Accelerating BERT inference via progressive word-vector elimination, in: International Conference on Machine Learning, PMLR, 2020, pp. 3690–3699.

[217] J. Kasai, N. Pappas, H. Peng, J. Cross, N.A. Smith, Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation, 2020, arXiv preprint arXiv:2006.10369.

[218] I. Beltagy, M.E. Peters, A. Cohan, Longformer: The long-document transformer, 2020, arXiv preprint arXiv:2004.05150.

[219] M. Zaheer, G. Guruganesh, K.A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al., Big bird: Transformers for longer sequences, Adv. Neural Inf. Process. Syst. 33 (2020) 17283–17297.

[220] M. Pagliardini, D. Paliotta, M. Jaggi, F. Fleuret, Faster causal attention over large sequences through sparse flash attention, 2023, arXiv preprint arXiv:2306.01160.

[221] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., An image is worth 16x16 words: Transformers for image recognition at scale, 2020, arXiv preprint arXiv:2010.11929.

[222] S. Anagnostidis, D. Pavllo, L. Biggio, L. Noci, A. Lucchi, T. Hoffmann, Dynamic context pruning for efficient and interpretable autoregressive transformers, 2023, arXiv preprint arXiv:2305.15805.

[223] X. Ma, C. Zhou, X. Kong, J. He, L. Gui, G. Neubig, J. May, L. Zettlemoyer, MEGA: Moving average equipped gated attention, 2022, arXiv preprint arXiv:2209.10655.

[224] S. Zuo, X. Liu, J. Jiao, D. Charles, E. Manavoglu, T. Zhao, J. Gao, Efficient long sequence modeling via state space augmented transformer, 2022, arXiv preprint arXiv:2212.08136.

[225] Q. Zhang, D. Ram, C. Hawkins, S. Zha, T. Zhao, Efficient long-range transformers: You need to attend more, but not necessarily at every layer, 2023, arXiv preprint arXiv:2310.12442.

[226] J. Ding, S. Ma, L. Dong, X. Zhang, S. Huang, W. Wang, N. Zheng, F. Wei, LongNet: Scaling transformers to 1,000,000,000 tokens, 2023, arXiv preprint arXiv:2307.02486.

[227] T. Dao, Flashattention-2: Faster attention with better parallelism and work partitioning, 2023, arXiv preprint arXiv:2307.08691.

[228] P. Zhang, G. Zeng, T. Wang, W. Lu, TinyLLaMA: An open-source small language model, 2024, arXiv:2401.02385.

[229] B. Chen, P. Li, B. Li, C. Li, L. Bai, C. Lin, M. Sun, J. Yan, W. Ouyang, PSViT: Better vision transformer via token pooling and attention sharing, 2021, arXiv preprint arXiv:2108.03428.

[230] Y. Li, Y. Lin, T. Xiao, J. Zhu, An efficient transformer decoder with compressed sub-layers, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, (15) 2021, pp. 13315–13323.

[231] K. Wu, Y. Zhang, B. Hu, T. Zhang, Speeding up transformer decoding via an attention refinement network, in: Proceedings of the 29th International Conference on Computational Linguistics, 2022, pp. 5109–5118.

[232] N. Shazeer, Fast transformer decoding: One write-head is all you need, 2019, arXiv preprint arXiv:1911.02150.

[233] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, S. Sanghai, GQA: Training generalized multi-query transformer models from multi-head checkpoints, 2023, arXiv preprint arXiv:2305.13245.

[234] H. Cao, C. Bao, C. Liu, H. Chen, K. Yin, H. Liu, Y. Liu, D. Jiang, X. Sun, Attention where it matters: Rethinking visual document understanding with selective region concentration, in: ICCV, 2023, pp. 19517–19527.

[235] T. Zhu, Y. Shi, Y. Zhang, Y. Wu, F. Mo, J.-Y. Nie, Collaboration and transition: Distilling item transitions into multi-query self-attention for sequential recommendation, 2023, arXiv preprint arXiv:2311.01056.

[236] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, et al., H _2 o: Heavy-hitter oracle for efficient generative inference of large language models, 2023, arXiv preprint arXiv:2306.14048.

[237] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, Y. You, Colossal-AI: A unified deep learning system for large-scale parallel training, in: Proceedings of the 52nd International Conference on Parallel Processing, 2023, pp. 766–775.

[238] G. Xiao, Y. Tian, B. Chen, S. Han, M. Lewis, Efficient streaming language models with attention sinks, 2023, arXiv preprint arXiv:2309.17453.

[239] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, 2022, arXiv preprint arXiv:2203.11171.
[240] S. Wang, B.Z. Li, M. Khabsa, H. Fang, H. Ma, Linformer: Self-attention with linear complexity, 2020, arXiv preprint arXiv:2006.04768.
[241] T. Schuster, A. Fisch, T. Jaakkola, R. Barzilay, Consistent accelerated inference via confident adaptive transformers, 2021, arXiv preprint arXiv:2104.08803.
[242] A. Katharopoulos, A. Vyas, N. Pappas, F. Fleuret, Transformers are rnns: Fast autoregressive transformers with linear attention, in: International Conference on Machine Learning, PMLR, 2020, pp. 5156–5165.
[243] Y. Li, Y. Yu, Q. Zhang, C. Liang, P. He, W. Chen, T. Zhao, LoSparse: Structured compression of large language models based on low-rank and sparse approximation, 2023, arXiv preprint arXiv:2306.11222.
[244] H. Xia, Z. Zheng, Y. Li, D. Zhuang, Z. Zhou, X. Qiu, Y. Li, W. Lin, S.L. Song, Flash-LLM: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity, 2023, arXiv preprint arXiv:2309.10285.
[245] Z. Xu, Z. Liu, B. Chen, Y. Tang, J. Wang, K. Zhou, X. Hu, A. Shrivastava, Compress then prompt: Improving accuracy-efficiency trade-off of llm inference with transferable prompt, 2023, arXiv preprint arXiv:2305.11186.
[246] Y. Song, Z. Mi, H. Xie, H. Chen, PowerInfer: Fast large language model serving with a consumer-grade GPU, 2023, arXiv preprint arXiv:2312.12456.
[247] A. Mishra, J. Albericio Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, P. Micikevicius, Accelerating sparse deep neural networks, 2021, arXiv preprint arXiv:2104.08378.
[248] E. Frantar, D. Alistarh, Massive language models can be accurately pruned in one-shot, 2023, arXiv preprint arXiv:2301.00774.
[249] M. Sun, Z. Liu, A. Bair, J.Z. Kolter, A simple and effective pruning approach for large language models, 2023, arXiv preprint arXiv:2306.11695.
[250] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, Y.T. Lee, Textbooks are all you need II: Phi-1.5 technical report, 2023, arXiv preprint arXiv:2309.05463.
[251] T. Valicenti, J. Vidal, R. Patnaik, Mini-GPTs: Efficient large language models through contextual pruning, 2023, arXiv preprint arXiv:2312.12682.
[252] X. Ma, G. Fang, X. Wang, LLM-pruner: On the structural pruning of large language models, 2023, arXiv preprint arXiv:2305.11627.
[253] E. Kurtic, E. Frantar, D. Alistarh, ZipLM: Hardware-aware structured pruning of language models, 2023, arXiv preprint arXiv:2302.04089.
[254] M. Santacroce, Z. Wen, Y. Shen, Y. Li, What matters in the structured pruning of generative language models? 2023, arXiv preprint arXiv:2302.03773.
[255] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re, et al., Deja Vu: Contextual sparsity for efficient llms at inference time, in: International Conference on Machine Learning, PMLR, 2023, pp. 22137–22176.
[256] M. Xia, T. Gao, Z. Zeng, D. Chen, Sheared LLaMA: Accelerating language model pre-training via structured pruning, 2023, arXiv preprint arXiv:2310.06694.
[257] S. Guo, J. Xu, L.L. Zhang, M. Yang, Compresso: Structured pruning with collaborative prompting learns compact large language models, 2023, arXiv preprint arXiv:2310.05015.
[258] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, V. Chandra, LLM-QAT: Data-free quantization aware training for large language models, 2023, arXiv:2305.17888.
[259] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, QLoRA: Efficient finetuning of quantized LLMs, 2023, arXiv:2305.14314.
[260] J. Kim, J.H. Lee, S. Kim, J. Park, K.M. Yoo, S.J. Kwon, D. Lee, Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization, 2023, arXiv:2305.14152.
[261] Y. Bondarenko, R.D. Chiaro, M. Nagel, Low-rank quantization-aware training for LLMs, 2024, arXiv:2406.06385.
[262] M. Chen, W. Shao, P. Xu, J. Wang, P. Gao, K. Zhang, Y. Qiao, P. Luo, EfficientQAT: Efficient quantization-aware training for large language models, 2024, arXiv:2407.11062.
[263] R. Jin, J. Du, W. Huang, W. Liu, J. Luan, B. Wang, D. Xiong, A comprehensive evaluation of quantization strategies for large language models, 2024, arXiv preprint arXiv:2402.16775.
[264] E. Frantar, S. Ashkboos, T. Hoefler, D. Alistarh, GPTQ: Accurate post-training quantization for generative pre-trained transformers, 2023, arXiv:2210.17323.
[265] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, S. Han, AWQ: Activation-aware weight quantization for LLM compression and acceleration, 2024, arXiv:2306.00978.
[266] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, S. Han, SmoothQuant: Accurate and efficient post-training quantization for large language models, in: Proceedings of the 40th International Conference on Machine Learning, 2023.
[267] T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, D. Alistarh, SPQR: A sparse-quantized representation for near-lossless llm weight compression, 2023, arXiv preprint arXiv:2306.03078.
[268] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, S. Han, QServe: W4a8kv4 quantization and system Co-design for efficient LLM serving, 2024, arXiv:2405.04532.
[269] X. Zhu, J. Li, Y. Liu, C. Ma, W. Wang, A survey on model compression for large language models, 2024, arXiv:2308.07633.
[270] S. Han, Tiny-chat, 2024, https://hanlab.mit.edu/blog/tinychat-vlm.
[271] E. Frantar, Autogptq, 2023, https://github.com/AUTOGPTQ/AutoGPTQ?tab=readme-ov-file.
[272] G. Gerganov, Llama.cpp, 2024, https://github.com/ggerganov/llama.cpp.
[273] NVIDIA, Tensorrt-LLM, 2023, https://github.com/NVIDIA/TensorRT-LLM.
[274] Y. Gu, L. Dong, F. Wei, M. Huang, Knowledge distillation of large language models, 2023, arXiv preprint arXiv:2306.08543.
[275] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, Q. Liu, Tinybert: Distilling bert for natural language understanding, 2019, arXiv preprint arXiv:1909.10351.
[276] R. Agarwal, N. Vieillard, P. Stanczyk, S. Ramos, M. Geist, O. Bachem, GKD: Generalized knowledge distillation for auto-regressive sequence models, 2023, arXiv preprint arXiv:2306.13649.
[277] S. Sun, Y. Cheng, Z. Gan, J. Liu, Patient knowledge distillation for bert model compression, 2019, arXiv preprint arXiv:1908.09355.
[278] X. Li, L. Lin, S. Wang, C. Qian, Unlock the power: Competitive distillation for multi-modal large language models, 2023, arXiv preprint arXiv:2311.08213.
[279] L. Li, Y. Zhang, L. Chen, Prompt distillation for efficient llm-based recommendation, in: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, 2023, pp. 1348–1357.
[280] L. Tunstall, E. Beeching, N. Lambert, N. Rajani, K. Rasul, Y. Belkada, S. Huang, L. von Werra, C. Fourrier, N. Habib, et al., Zephyr: Direct distillation of LM alignment, 2023, arXiv preprint arXiv:2310.16944.
[281] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J.E. Gonzalez, et al., Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality, 2023, See https://vicuna.lmsys.org. (Accessed 14 April 2023).
[282] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, D. Jiang, WizardLM: Empowering large language models to follow complex instructions, 2023, arXiv preprint arXiv:2304.12244.
[283] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q.V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, Adv. Neural Inf. Process. Syst. 35 (2022) 24824–24837.
[284] L.C. Magister, J. Mallinson, J. Adamek, E. Malmi, A. Severyn, Teaching small language models to reason, 2022, arXiv preprint arXiv:2212.08410.
[285] N. Ho, L. Schmid, S.-Y. Yun, Large language models are reasoning teachers, 2022, arXiv preprint arXiv:2212.10071.
[286] Y. Fu, H. Peng, L. Ou, A. Sabharwal, T. Khot, Specializing smaller language models towards multi-step reasoning, 2023, arXiv preprint arXiv:2301.12726.
[287] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, T. Pfister, Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes, 2023, arXiv preprint arXiv:2305.02301.
[288] S. Wadhwa, S. Amir, B.C. Wallace, Revisiting relation extraction in the era of large language models, 2023, arXiv preprint arXiv:2305.05003.
[289] P. Wang, Z. Wang, Z. Li, Y. Gao, B. Yin, X. Ren, SCOTT: Self-consistent chain-of-thought distillation, 2023, arXiv preprint arXiv:2305.01879.
[290] Y. Huang, Y. Chen, Z. Yu, K. McKeown, In-context learning distillation: Transferring few-shot learning ability of pre-trained language models, 2022, arXiv preprint arXiv:2212.10670.
[291] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. baker, M. Lai, A. Bolton, Y. Chen, T.P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, Nature 550 (2017) 354–359.
[292] W. Liang, G.A. Tadesse, D. Ho, L. Fei-Fei, M.A. Zaharia, C. Zhang, J. Zou, Advances, challenges and opportunities in creating data for trustworthy AI, Nat. Mach. Intell. 4 (2022) 669–677.
[293] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, X. Yi, A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability, Comp. Sci. Rev. 37 (2020) 100270.
[294] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, R. Ashmore, DeepConcolic: Testing and debugging deep neural networks, in: ICSE-Companion, 2019, pp. 111–114.
[295] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu, J. Zhao, DeepMutation++: A mutation testing framework for deep learning systems, in: 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE, 2019, pp. 1158–1161.
[296] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, R. Long, Technical report on the CleverHans v2.1.0 adversarial examples library, 2018, arXiv preprint arXiv:1610.00768.
[297] S. Ni, J. Li, H. Kao, DropAttack: A masked weight adversarial training method to improve generalization of neural networks, 2021, CoRR, arXiv:2108.12805.
[298] B. Schölkopf, Causality for machine learning, 2019, CoRR, arXiv:1911.10500, URL http://arxiv.org/abs/1911.10500.

[299] K. Sun, Q. Zhao, X. Wang, Using knowledge inference to suppress the lamp disturbance for fire detection, J. Saf. Sci. Resil. 2 (3) (2021) 124–130.
[300] Y. Liu, Y. Yao, J.-F. Ton, X. Zhang, R.G.H. Cheng, Y. Klochkov, M.F. Taufiq, H. Li, Trustworthy LLMs: A survey and guideline for evaluating large language models' alignment, 2023, arXiv preprint arXiv:2308.05374.
[301] J. Xu, J. Chen, S. You, Z. Xiao, Y. Yang, J. Lu, Robustness of deep learning models on graphs: A survey, AI Open 2 (2021) 69–78.
[302] C. Buckner, R. Miikkulainen, S. Forrest, Milano, AI reflections in 2021, Nat. Mach. Intell. 4 (1) (2022) 5–10.
[303] Y. He, Z. Shen, P. Cui, Towards non-I.I.D. image classification: A dataset and baselines, Pattern Recognit. 110 (2021) 107383, URL https://www.sciencedirect.com/science/article/pii/S0031320320301862.
[304] K. Sun, X. Ma, P. Liu, Q. Zhao, MPSN: Motion-aware pseudo-siamese network for indoor video head detection in buildings, Build. Environ. 222 (2022) 109354.
[305] S. Wu, G. Wang, P. Tang, F. Chen, L. Shi, Convolution with even-sized kernels and symmetric padding, Adv. Neural Inf. Process. Syst. 32 (2019).
[306] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.
[307] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al., Searching for mobilenetv3, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1314–1324.
[308] X. Zhang, X. Zhou, M. Lin, J. Sun, Shufflenet: An extremely efficient convolutional neural network for mobile devices, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856.
[309] N. Ma, X. Zhang, H.-T. Zheng, J. Sun, Shufflenet v2: Practical guidelines for efficient cnn architecture design, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 116–131.
[310] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, H. Hajishirzi, ESPNet: Efficient spatial pyramid of dilated convolutions for semantic segmentation, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 552–568.
[311] S. Mehta, M. Rastegari, L. Shapiro, H. Hajishirzi, ESPNetv2: A light-weight, power efficient, and general purpose convolutional neural network, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 9190–9200.
[312] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, C. Xu, GhostNet: More features from cheap operations, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1580–1589.
[313] S. Byna, X.-H. Sun, W. Gropp, R. Thakur, Predicting memory-access cost based on data-access patterns, in: 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No. 04EX935), IEEE, 2004, pp. 327–336.
[314] C.-Y. Wang, H.-Y.M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, I.-H. Yeh, CSPNet: A new backbone that can enhance learning capability of CNN, in: CVPR, 2020, pp. 390–391.
[315] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han, et al., MCUNet: Tiny deep learning on IoT devices, Adv. Neural Inf. Process. Syst. 33 (2020) 11711–11722.
[316] S. Liu, T. Chen, X. Chen, X. Chen, Q. Xiao, B. Wu, T. Kärkkäinen, M. Pechenizkiy, D. Mocanu, Z. Wang, More convnets in the 2020s: Scaling up kernels beyond 51 × 51 using sparsity, 2022, arXiv preprint arXiv:2207.03620.
[317] A. Wang, H. Chen, Z. Lin, H. Pu, G. Ding, RepViT: Revisiting mobile CNN from ViT perspective, 2023, arXiv preprint arXiv:2307.09283.

Kailai Sun received the B.S. degree from the Beijing University of Posts and Telecommunications, the Ph.D. degree from the Department of Automation, Tsinghua University, in 2023. He is currently a research fellow with the College of Design and Engineering, National University of Singapore. His research interests include computer vision, machine learning, mobile applications, AI for Science, smart buildings.

Xinwei Wang received the B.S. degree from the Tsinghua University, the M.S. degree from the Department of Automation, Tsinghua University, in 2024. His research interests include computer vision, machine learning, network slimming.

Xi Miao is currently pursuing the B.S. degree with the Department of

Mathematics at Pennsylvania State University. His research interests include AI, machine learning for vision.

Qianchuan Zhao received the B.E. degree in automatic control, the B.S. degree in applied mathematics, in 1992, and the M.S. and Ph.D. degrees in control theory and its applications from Tsinghua University, Beijing, China, in 1996. He was a Visiting Scholar with Carnegie Mellon University, Pittsburgh, PA, USA, in 2000; and Harvard University, Cambridge, MA, USA, in 2002. He was a Visiting Professor with Cornell University, Ithaca, NY, USA, in 2006. He is currently a Professor and the Director of the Center for Intelligent and Networked Systems (CFINS), Department of Automation, and the BNRist, Tsinghua University. He has published more than 100 research papers in peer reviewed journals and conferences. His current research interests include the control and optimization of complex networked systems with applications in smart buildings, smart grid, and manufacturing automation. Prof. Zhao received the 2009 and 2018 China National Nature Science Awards and the 2014 National Science Foundation for Distinguished Young Scholars of China. He is currently the Editor-in-Chief of the Results in Control and Optimization (RICO), an Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING (T-ASE), and an Associate Editor of the Journal of Optimization Theory and Applications.