Artificial Intelligence Review (2022) 55:6125–6183
https://doi.org/10.1007/s10462-022-10141-4

AI on the edge: a comprehensive review

Weixing Su[1] · Linfeng Li[2] · Fang Liu[3] · Maowei He[1] · Xiaodan Liang[1]

Published online: 21 March 2022

Abstract
With the advent of the Internet of Everything, the proliferation of data has put a huge burden on data centers and network bandwidth. To ease the pressure on data centers, edge computing, a new computing paradigm, is gradually gaining attention. Meanwhile, artificial intelligence services based on deep learning are also thriving. However, such intelligent services are usually deployed in data centers, which cause high latency. The combination of edge computing and artificial intelligence provides an effective solution to this problem. This new intelligence paradigm is called edge intelligence. In this paper, we focus on edge training and edge inference, the prior training models using local data at the resource-constrained edge devices. The latter deploying models at the edge devices through model compression and inference acceleration. This paper provides a comprehensive survey of existing architectures, technologies, frameworks and implementations in these two areas, and discusses existing challenges, possible solutions and future directions. We believe that this survey will make more researchers aware of edge intelligence.

Keywords Edge computing · Deep learning · Edge intelligence · Artificial intelligence · Edge devices

✉ Fang Liu
lf@tiangong.edu.cn

Weixing Su
suweixing@tiangong.edu.cn

Linfeng Li
llftjpu@163.com

Maowei He
hemaowei@hotmail.com

Xiaodan Liang
lxdtjpu@163.com

[1] School of Computer Science and Technology, Tiangong University, Tianjin 300387, China

[2] School of Artificial Intelligence, Tiangong University, Tianjin 300387, China

[3] School of Software, Tiangong University, Tianjin 300387, China

1 Introduction

With the constant development of algorithms, big data and computing power, artificial intelligence (AI) is flourishing in today's era. Among them, deep learning (DL) (Hinton et al. 2006) has yielded breakthrough achievements in various fields, from computer vision (Krizhevsky et al. 2012) to natural language processing (Bengio et al. 2003). The rapid development in these fields has produced many DL-based smart application services, which bring great convenience to our lives. However, existing smart application services are often computationally intensive, and these DL-based services are usually deployed in data centers. Such cloud-based DL services have several drawbacks: (1) computation tasks are deployed in data centers, which can impose a huge burden on network bandwidth and data centers; (2) it cannot provide low-latency services; (3) data is uploaded to the cloud, which can be a potential threat to user privacy.

As the extension of cloud computing, edge computing (Shi et al. 2016) can effectively alleviate the issues mentioned above. It relieves the pressure on network bandwidth and data centers by sinking part of the computation to the edge. Compared to cloud computing, edge computing has the following advantages: (1) computation is performed locally or at the edge servers, lowering latency; (2) data does not need to be uploaded to the cloud, reducing bandwidth consumption; (3) data is stored at the edge, which protects user privacy. Edge computing offers an effective solution to the key challenges of AI. In fact, the combination of edge computing and AI has formed a new intelligence paradigm, edge intelligence. Edge intelligence deploys a portion of DL services to the edge, improving the speed of data processing, avoiding user privacy leakage and reducing bandwidth consumption.

As far as we know, there exists some efforts related to our work (Zhou et al. 2019b; Xu et al. 2020; Wang et al. 2020; Tang et al. 2020; Deng et al. 2020; Yang et al. 2019; Kholod et al. 2021; Yang et al. 2020a). Specifically, Tang et al. (2020) provided a comprehensive survey on communication-efficient distributed training. They classified distributed training architectures into three types from the system-level, which are also involved in our work. Yang et al. (2019) provided a comprehensive survey about federated learning (FL), which they classified as horizontal FL, vertical FL, and federated transfer learning. FL is also an important component of our survey. Kholod et al. (2021) investigated open-source frameworks for FL. They conducted experiments on three datasets and evaluated the applicability of the FL frameworks to Internet of Things (IoT) systems in terms of five dimensions: ease of use and deployment, development, analysis capabilities, accuracy, and performance. Yang et al. (2020a) presents the main contributions of 35 papers on edge intelligence. Some of these papers are also involved in our survey. The survey of Wang et al. (2020), Deng et al. (2020) consists of two parts: (1) how to train and deploy models in resource-constrained edge devices; and (2) how to utilize AI to solve problems in edge computing. The former is the focus of our survey, and the latter is outside the scope of our survey. In addition, their survey was conducted from a macro perspective. Compared to their work, we provide more detailed information and the latest research.

To the best of our knowledge, Zhou et al. (2019b) and Xu et al. (2020) are the two most relevant reviews to our survey. Xu et al. (2020) investigated the works on edge intelligence from four perspectives: edge caching, edge training, edge inference, and edge offloading. The efforts of these four areas are also involved in our survey. The difference is that we focus on the efforts on edge training and edge inference. For edge caching as well as edge offloading, we focus on their applications to edge intelligence. In addition to this,

Table 1 The comparison of related surveys

Related work	Domain	Content	Differences
Tang et al. (2020)	Distributed learning	Communication-efficient distributed learning algorithms are introduced	It is not specifically for AI on the edge
Yang et al. (2019)	Federated learning	Three forms of federated learning are introduced	It is not specifically for AI on the edge
Kholod et al. (2021)	Federated learning	FL open-source framework's applicability to IoT systems is evaluated	We investigate other areas of AI on the edge such as model compression, inference acceleration, etc
Yang et al. (2020a)	Edge intelligence	The main contributions of 35 papers on edge intelligence are introduced	We provide information on other subfields of AI on the edge (e.g., model compression) and some other details (e.g., open-source frameworks)
Wang et al. (2020)	Edge intelligence and intelligent edge	Edge intelligence and intelligent edge are investigated from a macro perspective	We provide more detailed information (technologies development, training architectures, inference architectures, etc.) about edge training and edge inference
Deng et al. (2020)	Edge intelligence and intelligent edge	AI for edge and AI on edge are investigated from a macro perspective	We provide more detailed information (technologies development, training architectures, inference architectures, etc.) about edge training and edge inference
Zhou et al. (2019b)	Edge intelligence	Edge training and edge inference are investigated from a macro perspective	We provide more detailed information (open-source frameworks, AI implementations on the edge, etc.) and the latest research on edge training and edge inference
Xu et al. (2020)	Edge intelligence	Edge training, edge inference, edge caching and edge offloading are investigated from a microscopic perspective	We provide more detailed information (training architectures, inference architectures, open-source frameworks, etc.) and the latest research on edge training and edge inference
Ours	Edge intelligence	The latest research and detailed information on the architectures, technical developments, available frameworks and implementations for edge training and edge inference are investigated.	–

we provide a more detailed introduction and the latest research. Zhou et al. (2019b) introduced the works on edge training and edge inference from a macroscopic perspective. Differently, our investigation is conducted from a microscopic perspective. Moreover, (Zhou et al. 2019b) was published in 2019. Therefore, we investigated the most recent studies and provided more detailed information. We compare these efforts in Table 1.

In this paper, we mainly focus on edge training and edge inference. The former uses federated learning to train models on edge devices. The latter uses techniques such as model compression and inference acceleration to deploy models on edge devices. The remainder of this paper is organized as follows. We introduce the foundations of edge computing and AI in Sects. 2 and 3, respectively. In Sects. 4 and 5, we provide a detailed introduction, discussion, and analysis of the techniques, architectures, open-source frameworks and developments in edge training and edge inference, separately. The implementations of edge intelligence is presented in Sect. 6. Finally, we provide open challenges, possible solutions and future directions for edge intelligence in Sect. 7.

2 The foundations of edge computing

In this section, we introduce some fundamental concepts of edge computing, which consists of four parts: cloud computing, fog computing, edge computing, and cloud-edge collaboration.

(1) Cloud computing: the National Institute of Standards and Technology (NIST) defined cloud computing as a ubiquitous, easily accessible, and on-demand accessed shared pool of computing resources that can be rapidly deployed with minimal management (Mell and Grance 2011). Cloud computing brings together numerous computing resources and manages them automatically through software. Cloud computing provides fast and secure computing and data storage services that allow everyone to utilize the huge computing resources on the network.

(2) Fog computing: Bonomi et al. (2012) first introduced fog computing, defined it as a highly virtualized computing platform that migrates computing tasks from the cloud to network edge for execution. Different from cloud computing, fog computing is not composed of powerful servers, but of weaker and more decentralized computing nodes. It is an IoT-oriented distributed computing infrastructure that can significantly reduce latency by sinking computation power and data analysis to the edge of the network.

(3) Edge computing: edge computing is located at the edge of the network that is close to devices. The edge can be any entity from the data source to the data center and provides services such as computation, storage, and data analysis to the devices. It is worth noting that the edge is not the device itself. Although both edge computing and fog computing move computation and storage to the edge of the network, there are certain differences between the two computation paradigms. Group OCAW et al. (2017) pointed out that fog computing provides computation, storage, and control services for all requests between the cloud and devices, while edge computing is only for edge devices.

(4) Cloud-edge collaboration: cloud computing can provide powerful support for computationally intensive tasks such as DL, however it cannot provide low-latency services. Although edge computing can effectively alleviate this problem, it is difficult to run computationally intensive tasks on resource-constrained devices. Collaboration between cloud and edge can provide low-latency DL services. For example, Huang et al. (2019) split deep neural network (DNN) between cloud, edge and devices, allow-

ing a portion of the computation to be performed at the edge or devices, thus achieving trade-offs between latency, bandwidth consumption and device energy consumption.

3 The foundations of AI

In this section, we introduce some subfields of machine learning and focus on deep neural network.

3.1 Machine learning

Machine learning is a technique that uses algorithms to parse data and make classifications and predictions about real-world things. Machine learning comes directly from the early days of the AI field. Its traditional algorithms include decision trees, clustering, Bayesian classification, support vector machines (SVM), Adaboost, etc. Here we introduce several subfields of machine learning: incremental learning, active learning, reinforcement learning, deep learning and deep reinforcement learning.

(1) Incremental learning: general machine learning models forget what they have learned while they acquire new knowledge, which is known as "catastrophic forgetting". Models can be retrained with the help of data centers. However, this can be time- and energy-consuming. Incremental learning can be an efficient solution to this problem. Models obtained by training with incremental learning can perform well on both old and new tasks. It is worth noting that the concept, incremental learning, is similar to continuous learning and lifelong learning.
(2) Active learning: supervised learning requires a complete annotated dataset to train the model. Data annotation is costly when the dataset is extremely large. Active learning is an effective way to alleviate this problem. The core idea of active learning is to allow the learning algorithm to proactively propose annotation requests and hand over some relatively difficult annotation tasks to human experts for further annotation. In this way, the costs of data annotation can be effectively decreased.
(3) Reinforcement learning: reinforcement learning emphasizes how to maximize benefits by taking actions based on the environment. In contrast to supervised learning, reinforcement learning (Bellman 1953) does not require any data. It acquires learning information and updates model parameters by accepting the feedback from the environment. Reinforcement learning is often used in applications such as Go and games, e.g., AlphaGo (Silver et al. 2016) is an implementation based on reinforcement learning.
(4) Deep learning: as one of the main driving engines of AI development, DL has attracted wide attention of researchers since AlexNet (Krizhevsky et al. 2012) won the ImageNet competition, after which more excellent network models were proposed, such as VGG (Simonyan and Zisserman 2014), GoogleNet (Szegedy et al. 2015). DL has made significant breakthroughs in many fields such as data mining, machine translation, face recognition, natural language processing, etc. It has solved a number of complicated pattern recognition problems and enabled the rapid development of AI.
(5) Deep reinforcement learning: DL has a high perceptual ability but lacks a certain degree of decision-making ability. Whereas, reinforcement learning possesses decision-making ability but cannot perceive the target well. Deep reinforcement learning (Mnih et al. 2013) integrates the perceptual ability of DL and the decision-making ability of

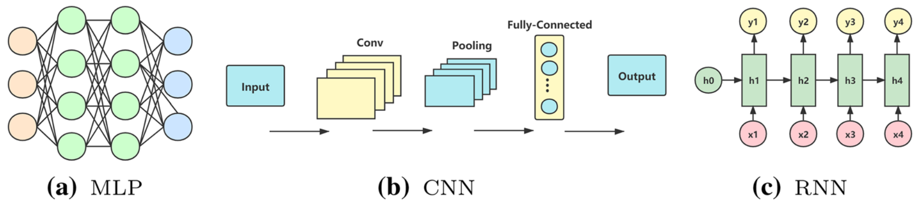


Fig. 1 Three types of common neural networks

reinforcement learning. It can control directly based on the input image. This approach is closer to the pattern of human thought.

3.2 Deep neural network

DNN can be comprehended as a neural network structure that includes multiple hidden layers. Here, we focus on three structures of DNN: multilayer perceptron (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).

The earliest neural network is perceptron, which has an input layer, an output layer and a hidden layer. The input feature vector is transformed through the hidden layer to the output layer, and the results are obtained in the output layer. To fit more complex functions, (McCulloch and Pitts 1943) proposed MLP. As shown in Fig. 1a, MLP is a series of fully-connected layers. The problem of fully connected DNNs is the excessive number of parameters. To alleviate this problem, LeCun et al. (1989) introduced convolution kernels in CNN. The weights of the same convolution kernel are shared. The image still retains its original position relationship after being convolved. In this way, CNN effectively reduces the number of parameters. CNN is mainly applied to computer vision, but it can also be applied to natural language processing (Zhang and Wallace 2015). RNN can represent the changes of time-series, and it is widely applied in natural language processing. In order to better handle time-series tasks, a series of RNN variants had been proposed, such as LSTM (Hochreiter and Schmidhuber 1997), Transformer (Vaswani et al. 2017). Recently, a number of researchers have paid attention to the application of Transformer in computer vision (Dosovitskiy et al. 2020; Touvron et al. 2020). All these works obtained good performance.

4 Edge training

Cloud-based DL requires a complete dataset in the data center for training. However, training tasks in the edge environment are mainly done through the collaboration of multiple participants. Such distributed learning is known as FL, which can effectively save bandwidth and protects user privacy.

There are some differences between FL and distributed learning (Table 2). First, the purpose of distributed learning is to process large amounts of data in parallel, while the purpose of FL is to train models locally and protect user privacy. Second, traditional distributed learning usually works with independent and identically distributed (IID) data, while FL works with non-independent and identically distributed (Non-IID) data. Finally, traditional distributed learning uses high-performance compute nodes for

Table 2 The difference between distributed learning and federated learning

	Distributed learning	Federated learning
Aims	Processing large amounts of data in parallel	Training models locally and protecting user privacy
Data	IID data with almost the same size	Non-IID data; the amount of data varies greatly from client to client
Nodes	High performance computing nodes	Resource-constrained edge nodes

training, while FL uses resource-constrained edge nodes for training. In this section, we focus on training architectures of distributed learning and the development of FL.

4.1 Training architecture

(1) Parameter server: in the parameter server (PS) architecture (Smola and Narayanamurthy 2010), there are two roles, the server and working nodes. As shown in Fig. 2a, first, the working nodes train the model and update the parameters locally. Then the updated parameters are sent to the server. Finally, the server aggregates the updates to get a global model and distributes the parameters to each node. The main challenges of the PS architecture are communication bottlenecks and single-point-of-failure.

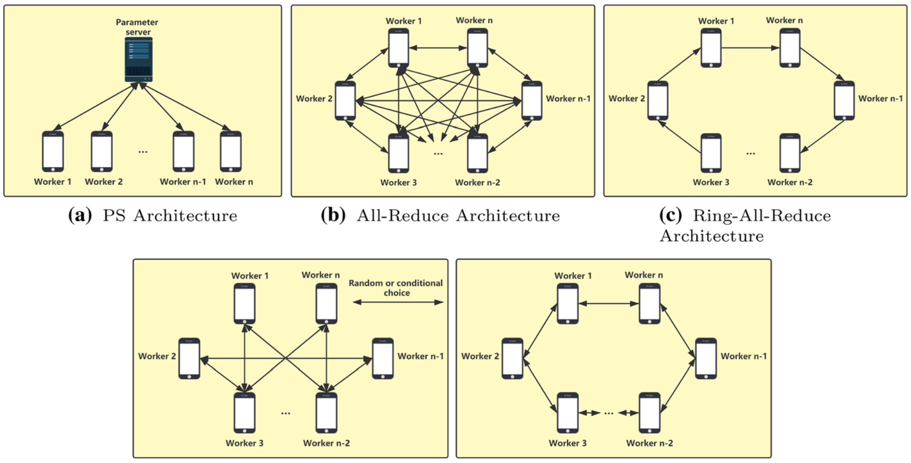
(2) All-reduce: to avoid single-point-of-failure, All-Reduce is an optional solution (Patarasuk and Yuan 2009). As shown in Fig. 2b, All-Reduce removes the PS. Each working node communicates with other working nodes and gets updates from all the nodes. Therefore, each node has a consistent global model. The collective communication pattern of All-Reduce prevents it from being utilized for asynchronous communication. Moreover, this solution results in significant bandwidth consumption due to frequent communication. In order to reduce bandwidth consumption, Gibiansky (2017) proposed Ring-All-Reduce. As shown in Fig. 2c, they constructed the working nodes into a ring structure, where each node delivers updates to the next node in a clockwise direction. At the same time, it also receives updates from the last node.

(3) Gossip: in contrast to the PS architecture, the gossip architecture has neither PS nor a consistent global model. As shown in Fig. 2d, in this architecture, the working nodes randomly (Blot et al. 2016) or conditionally (Roy et al. 2019) select several nodes to exchange updates at each iteration. Therefore, in each iteration, all the local models are different from each other. However, it guarantees that the final model is consistent.

(4) Neighbor: similar to the gossip architecture, the neighbor architecture does not have a consistent global model either. As shown in Fig. 2e, in each iteration, each working node communicates only with its neighbor nodes. Therefore, the model on each node is changed in each iteration. Of course, it also ensures the consistency of the final model.

4.2 Federated learning

FL was proposed by Google. It is a distributed learning that takes place in mobile edge systems. The most common application scenario of FL (McMahan et al. 2017) has the following characteristics: (1) network instability: edge devices are often disconnected; (2) data imbalance: the data volume may be significantly different between edge devices; (3) Non-IID data: the data distribution of each edge device cannot represent the total data



(a) PS Architecture **(b)** All-Reduce Architecture **(c)** Ring-All-Reduce Architecture

(d) Gossip Architecture **(e)** Neighbor Architecture

Fig. 2 Five types of training architectures

distribution due to the different preferences among users; (4) large number of devices: there are a huge number of devices in the edge.

FL with PS architecture appeared relatively early. Then, FL with peer-to-peer architecture emerged, which reduces bandwidth consumption and avoids the single-point-of-failure compared to FL with PS architecture. We classify these efforts into four categories: (1) communication-efficient FL, (2) privacy preserving FL, (3) FL with Non-IID data, and (4) Decentralized FL. The major efforts of FL are summarized in Table 3.

4.2.1 Communication-efficient FL

As shown in Fig. 3, for FL with PS architecture, edge devices send updates to the PS and get the global parameters from the PS. However, In the edge environment, network bandwidth resources are limited. Therefore, communication optimization is essential. We categorize the efforts of communication optimization into two types: (1) communication frequency optimization and (2) communication costs optimization. Among them, the optimization of communication frequency avoids frequent update exchanges by controlling communication rounds. The optimization of communication costs reduce the size of the updates in each communication round.

Communication frequency optimization McMahan et al. (2017) first proposed FL and presented the Federated Averaging (FedAvg). They reduced the number of communication rounds by adding additional computations on local devices. Compared to synchronized stochastic gradient descent, FedAvg reduces communication rounds by a factor of 10–100. Wang et al. (2019) proposed a control algorithm which can dynamically adjust the global aggregation frequency to minimize the learning loss under a fixed resource budget. Reisizadeh et al. (2020) proposed a periodic average and quantization (FedPAQ) algorithm. FedPAQ allows clients to perform gradient updates locally, capturing the updates of devices in phases and sending them to the server. Moreover, the information sent to the server is simply a quantified version of the local information. The periodic average reduces the number of communication rounds and the quantization decreases the costs of each communication round.

Table 3 The major efforts of Federated Learning

Related work	Objective	Algorithm/framework	Highlight
McMahan et al. (2017)	Communication frequency	FedAvg	Adding additional computation locally
Wang et al. (2019)	Communication frequency	A control algorithm	Adjusting the global aggregation frequency dynamically
Reisizadeh et al. (2020)	Communication frequency and cost	FedPAQ	Periodic average and quantization
AbdulRahman et al. (2020)	Communication frequency	FedMCCS	Evaluating the ability of clients to perform FL tasks
Lin et al. (2017b)	Communication cost	DGC	No damage to model accuracy
Konečný et al. (2016)	Communication cost	Structured updates and sketched updates	Reducing the uplink communication bandwidth
Caldas et al. (2018)	Communication cost	Federated dropout	Focusing server-to-client communication
Jiang et al. (2019)	Communication cost	PruneFL	Including initial pruning and distributed pruning
Chen et al. (2019)	Communication cost	Asynchronous learning and temporally weighted aggregation	Different aggregation frequencies for deep and shallow layers
Aono et al. (2017)	Protecting privacy from PS	–	Applying homomorphic encryption
Geyer et al. (2017)	Protecting privacy from PS	–	Applying differential privacy
Bonawitz et al. (2017)	Protecting privacy from PS	A Secure aggregation protocol	Applying secure multi-party computation
Truex et al. (2019)	Protecting privacy from PS	–	Combining differential privacy with secure multi-party computation
Wei et al. (2020)	Protecting privacy from PS	NbAFL	Providing theoretical analysis for FL with differential privacy
Kim et al. (2019)	Protecting privacy from working nodes	BlockFL	Introducing incentives to attract more clients to engage in training
Weng et al. (2021)	Protecting privacy from working nodes	DeepChain	Providing data confidentiality, computational auditability and incentives
Zhao et al. (2020)	Protecting privacy from working nodes	–	Combining differential privacy with blockchain
Zhao et al. (2018a)	Processing Non-IID data	–	Creating a subset of data that is shared globally for training on Non-IID data
Yoshida et al. (2020)	Processing Non-IID data	Hybrid-FL	Assuming that a few clients allow to upload their data to the server and form an approximate IID dataset
Zhang et al. (2021b)	Processing Non-IID data	CSFedAvg	Selecting clients with lower degree of Non-IID data to train the model

Table 3 (continued)

Related work	Objective	Algorithm/framework	Highlight
Duan et al. (2020)	Processing Non-IID data	–	Mitigating the global data imbalance and local data imbalance
Lalitha et al. (2019)	Decentralized FL	–	Decentralized FL with neighbor architecture
Lu et al. (2020)	Communication frequency	–	Decentralized FL with neighbor architecture
Hu et al. (2019)	Communication cost	A segmented gossip approach	Decentralized FL with Gossip architecture
Roy et al. (2019)	Decentralized FL	BrainTorrent	Decentralized FL with Gossip architecture
Savazzi et al. (2020)	Decentralized FL	Consensus-based FedAvg algorithms	Decentralized FL with Gossip architecture

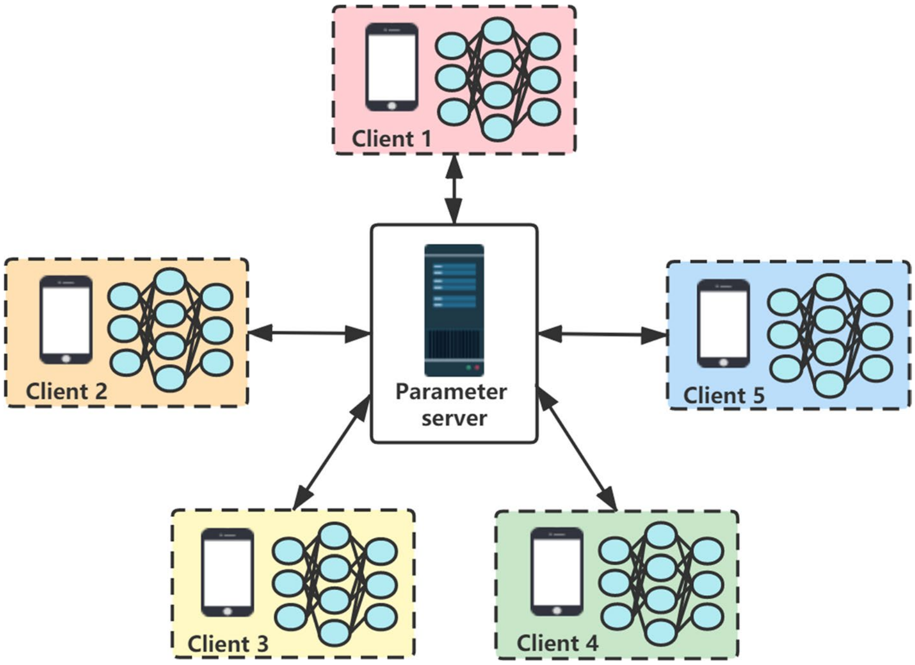


Fig. 3 FL with PS Architecture

The aforementioned works randomly selected clients during each training round, which may lead to unreliable client selection and cause unnecessary bandwidth consumption. AbdulRahman et al. (2020) maximizes the number of users involved in training and reduced the number of discarded communication rounds by evaluating the ability of users to perform FL tasks from multiple perspectives, including CPU, memory, energy and time.

Communication costs optimization In contrast to communication frequency optimization, the optimization of communication costs focuses on reducing the size of the updates in each round of communication. Lin et al. (2017b) discovered that in distributed stochastic gradient descent, up to 99% of the gradient exchanges are unnecessary. Therefore, communication costs can be decreased by reducing redundant gradient exchanges. To achieve this goal, gradient quantization and gradient sparsification are widely used. For example, Seide et al. (2014) used 1-bit quantization to reduce bandwidth consumption in stochastic gradient descent. Aji and Heafield (2017) sparsified the gradient updates by removing a certain percentage of the gradients based on the absolute value of the threshold, and used a global threshold with layer normalization. However, these works either added additional layer normalization or brought a loss of accuracy. Lin et al. (2017b) proposed deep gradient compression (DGC). Compared to the works mentioned above, DGC does not require extra layer normalization and uses momentum correction and local gradient clipping to ensure that the model accuracy is not degraded.

Konečnỳ et al. (2016) proposed two approaches for reducing the uplink communication bandwidth, structured updates and sketched updates. Structured updates refer to updates that can be learned from parameterized constrained space that uses fewer variables. Sketched updates refer to learning the complete model updates and then compressing them before sending to the server. Different from Konečnỳ et al. (2016), Caldas et al. (2018)

focused on server-to-client communication. In addition, inspired by the Dropout (Srivastava et al. 2014), they proposed the Federated Dropout, which allows each device locally trains sub-models. These sub-models are subsets of the global model. Thus, local updates can be considered as updates of the global model.

Jiang et al. (2019) proposed PruneFL, an adaptive distributed parameter pruning method based on FL. PruneFL includes initial pruning and distributed pruning. It also contains a low-complexity adaptive pruning method that achieves a trade-off between accuracy and computation time. Chen et al. (2019) proposed an asynchronous learning strategy and a temporally weighted aggregation strategy. In the asynchronous learning strategy, DNN layers are divided into shallow layers and deep layers. The parameters of shallow layers are updated more frequently than the parameters of deep layers, which reduce communication costs between the server and clients. In addition, the temporally weighted aggregation strategy can aggregate the local models more efficiently, improving the performance of the model.

4.2.2 Privacy preserving FL

Although FL protects user privacy to a certain degree, some works have demonstrated that it is possible to obtain certain user information from trained models. For example, Hitaj et al. (2017) designed a distributed DL attack based on generative adversarial network (GAN). Any user involved in collaborative training can use this attack to generate sensitive information about the victim's device. This attack is effective even when the client uses differential privacy to obfuscate parameters. The authors claimed that this attack is also effective for FL. Further, Melis et al. (2019) attacked to FL. They demonstrated that gradient updates can leak participants' training data and proposed active and passive inference attacks. Despite the fact that FL can average gradient updates, they still successfully exploited gradient updates for the attack.

In FL with PS architecture, privacy threats are from two main sources: (1) PS; and (2) malicious edge nodes. Early works mainly focused on privacy threats from PS (Aono et al. 2017; Geyer et al. 2017; Bonawitz et al. 2017). These works mainly protected user privacy through differential privacy, homomorphic encryption, and secure multi-party computation. Later, researchers used blockchain to eliminate privacy threats from PS and to mitigate privacy threats from malicious nodes to some extent.

Privacy threats from PS Shokri and Shmatikov (2015) proposed a privacy-preserving DL system that allows participants to train on their own datasets and selectively share a portion of the parameters. Sharing parameters can improve model accuracy, but brings privacy loss. The system can adjust the shared parameters to make a trade-off between privacy loss and model accuracy. However, Aono et al. (2017) found that even if only a portion of the parameters are shared, useful information about the users can still be obtained. Therefore, the authors used homomorphic encryption to protect the participants' information from being leaked to the server.

Geyer et al. (2017) applied differential privacy to FL for hiding the contribution of clients participating in training. The algorithm can achieve good convergence performance for a given privacy level, especially when a sufficient number of clients are involved in the training. Bonawitz et al. (2017) applied secure multi-party computation to FL and proposed an efficient secure aggregation protocol. This method makes it impossible to check individual updates for each client. Truex et al. (2019) proposed a FL algorithm that combines differential privacy and secure multi-party computation. In which, differential privacy protects user privacy and secure multi-party computation ensures that the information exchange without being protected by differential privacy is not disclosed.

The works mentioned above only proved the experimental results by simulation, but lacked theoretical analyses. Wei et al. (2020) provided a theoretical analysis of FL's convergence properties when differential privacy is used. The authors demonstrated theoretically that there is a trade-off between convergence performance and privacy preservation. They presented NbAFL, a framework that adds noise to the client's parameters before aggregation. In addition, the authors proposed a K-client stochastic scheduling strategy. They discovered that there is an optimal number of clients, K, for a given level of privacy, allowing the model to achieve the best convergence performance.

Privacy threats from edge nodes The works mentioned above focused on privacy threats from PS. However, dishonest clients can also impair training. For example, Bagdasaryan et al. (2020) demonstrated that dishonest clients can be used to attack FL through model replacement. In addition, clients participating in FL have absolute control over their local data. As a result, anomaly detection can't be used to discover malicious nodes. Researchers are already focusing on the application of blockchain in FL to protect privacy from malicious nodes.

Kim et al. (2019) proposed BlockFL, a blockchain-based architecture, which eliminates the privacy threats and single-point-of-failure posed by PS. Additionally, they introduced incentives to attract clients to engage in training and mitigate privacy threats from dishonest clients. Similarly, DeepChain Weng et al. (2021) can provide data confidentiality, computational auditability, and incentives to the parties involved in collaborative training. Taking a step further, differential privacy was introduced in the work of Zhao et al. (2020) for further protecting user privacy. Besides, the authors also proposed a normalization technique. This normalization technique outperforms batch normalization (BN) when the data features are under differential privacy protection.

4.2.3 FL with Non-IID data

Although there have been some studies focusing on the processing of unbalanced data distribution (Du et al. 2020, 2021), most of the existing studies on machine learning assume that the data distribution is identical. However, in FL, Non-IID data is even more of a non-negligible problem.

Li et al. (2019c) demonstrated that the heterogeneity of the data slows down the convergence rate. Similarly, Duan et al. (2019) proved that unbalanced distributed training data can reduce the accuracy of a FL-trained model. Although the FedAvg algorithm proposed by McMahan et al. (2017) is robust to a certain degree of Non-IID data, (Zhao et al. 2018a) proved that the accuracy of models trained with FedAvg is significantly decreased when the training dataset is highly-skewed Non-IID. They demonstrated that the accuracy reduction is caused by weight differences. The authors quantified the weight differences by Earth Mover's distances. In addition, they suggested creating a subset of data that is shared globally across devices to improve training on Non-IID data. However, users may refuse to install unknown data on their own devices due to security concerns.

Yoshida et al. (2020) proposed a new learning mechanism, Hybrid-FL, to solve the Non-IID data problem. They assumed that a few clients allow their data to be uploaded to the server in order to form an approximate IID dataset. The IID dataset is utilized to train a model. This model is aggregated with other models trained with Non-IID data. Zhang et al. (2021b) observed that there are weight differences between clients with different degrees of

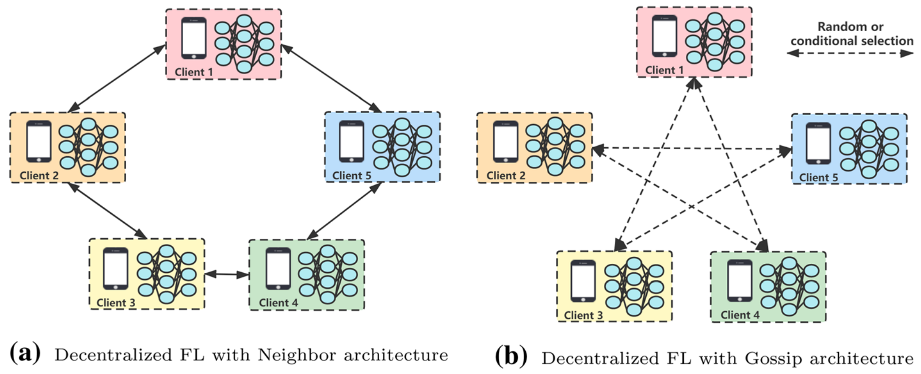

Client 1
Client 2
Client 3
Client 4
Client 5
(a) Decentralized FL with Neighbor architecture
Random or conditional selection
(b) Decentralized FL with Gossip architecture

Fig. 4 Decentralized FL

Non-IID data and those with IID data. Based on this, CSFedAvg, an efficient FL algorithm, was proposed. They identified the degree of Non-IID data that clients have by the weight differences and selected clients with lower degree of Non-IID data to train the model.

All the works mentioned above only considered the effect of local data distribution imbalance. Duan et al. (2019), Duan et al. (2020) mitigated the global data imbalance by data augmentation. For local data distribution imbalance, they introduced a mediator, which can rearrange clients training based on the Kullback–Leibler divergence of clients' data distribution. However, the data distribution of clients may disclose their privacy to some extent.

4.2.4 Decentralized FL

FL with PS architecture can protect user privacy to a certain degree. However, this architecture has two drawbacks: (1) it causes communication bottlenecks; (2) it is over-reliant upon the PS, which demands that all clients collectively choose a trusted PS. Moreover, the failure of PS can disrupt the entire training process. To alleviate these problems, some of the existing works discard PS, which can be achieved by blockchain (Weng et al. 2021; Kim et al. 2019; Lu et al. 2019; Zhao et al. 2020) and other communication methods (Lalitha et al. 2019; Hegedűs et al. 2019; Lu et al. 2020; Hu et al. 2019).

As shown in Fig. 4a, Lalitha et al. (2019) distributed all the nodes on a network. Each node communicated with its neighbors to train the model, in this way a decentralized FL was achieved. Hegedűs et al. (2019) systematically compared the aggregation costs of distributed learning for PS architecture and Gossip architecture and evaluated some additional techniques. They found that the performance of the two architectures are comparable.

Lu et al. (2020) and Hu et al. (2019) optimized the decentralized FL from the perspective of communication frequency and communication costs, respectively. In the work of Lu et al. (2020), clients update the gradients locally for certain rounds before communicating with their neighbors. By this way communication rounds can be effectively reduced. Hu et al. (2019) partitioned the model into mutually non-overlapping segments. Each node updated by aggregating local segments and segments from the other k nodes. During the training process, each node randomly selects a certain number of other nodes to transmit the model segments. Compared with transmitting the entire model, communication costs are significantly reduced.

4.2.5 Open-source FL frameworks

Several open-source frameworks have been proposed to facilitate the development of FL. Here we introduce four FL open-source frameworks, which are TensorFlow Federated Framework (TFF),[1] Federated AI Technology Enabler Framework (FATE),[2] Paddle Federated Learning Framework (PFL)[3] and PySyft Framework.[4]Kholod et al. (2021) provided a detailed introduction and evaluation of FL frameworks, which is helpful for those interested in FL frameworks.

TensorFlow federated framework TFF is an open-source framework proposed by Google for distributed machine learning. TFF was developed to promote open research and experimentation in FL. With TFF, developers can use models and data to simulate the FL algorithms it contains. TFF implements base classes for FedAvg and the Federated Stochastic Gradient Descent algorithm, which are simple implementations of federated evaluation.

TFF's interface can be divided into two layers: (1) FL API: this layer provides a high-level interface that enables developers to apply all the FL algorithms it contains to existing Tensorflow models. (2) Federated Core API: the core of this layer is a set of low-order interfaces, which allow new FL algorithms to be developed succinctly with a combination of Tensorflow and distributed communication operators. This layer is the basis for developers to build FL.

Federated AI technology enabler framework FATE is an open-source project initiated by the AI department of WeBank. To the best of our knowledge, FATE is the most popular FL framework. FATE builds secure computing protocols with multi-party secure computation and homomorphic encryption. Moreover, FATE supports secure computation for different kinds of machine learning including logistic regression, DL, transfer learning, etc.

FATE currently supports two deployment methods. (1) Naïve deployment, which includes stand-alone deployment and cluster deployment. Stand-alone deployment helps developers to develop quickly and test FATE. Cluster deployment is for big data scenarios. It is worth noting that migrating from stand-alone deployment to cluster deployment requires only a configuration file change and no algorithm changes. (2) KubeFATE deployment, with KubeFATE, FATE can be deployed via docker-compose or Kubernetes. Docker-compose is suitable for development or testing, while Kubernetes is suitable for large-scale deployments in production environments.

Paddle federated learning framework PFL is an open-source FL framework based on PaddlePaddle, both of which were proposed by Baidu. PFL can process horizontally partitioned and vertically partitioned data, and the algorithms for these two types of data partitioning are implemented in the paddle_fl package and the mpc package, respectively. In PFL, the components for defining FL tasks are as follows.

Components for compiling: (1) FL-strategy: defining the FL strategy; (2) User-Defined-Program: defining the machine learning model structure and training strategy; (3) Distributed-Config: defining the information about distributed training nodes; (4) FL-Job-Generator: generating FL-jobs for the server side and worker side. FL-jobs are distributed to the server side and worker side for joint training.

[1] https://github.com/tensorflow/federated.

[2] https://github.com/FederatedAI/FATE.

[3] https://github.com/PaddlePaddle/PaddleFL.

[4] https://github.com/OpenMined/PySyft.

Components for running: (1) FL-Server: a parameter server for model aggregation; (2) FL-Worker: worker nodes that train the model locally; (3) FL-Scheduler: scheduling worker nodes during training and deciding which workers can participate in training.

Notably, PFL requires at least 6GB of RAM and 100GB of hard disk space to run properly (Kholod et al. 2021), which prevents PFL from being used in IoT systems. In addition, PFL is based on PaddlePaddle, a DL framework proposed by Baidu. PaddlePaddle has a smaller user base than Tensorflow and Pytorch, which may be detrimental to the development of PFL.

PySyft framework PySyft is an open-source python framework that supports secure, private computation in DL. PySyft integrates FL, secure multi-party computation and differential privacy into different DL frameworks such as PyTorch, Keras or TensorFlow.

In order to extend PySyft to implement FL on mobile or edge devices, some other libraries are necessary such as KotlinSyft (Android) and SwiftSyft (iOS). These libraries belong to the Syft ecosystem. However, these libraries alone cannot connect user data to the real world. The Syft ecosystem can be supported by the Grid ecosystem, which focuses on the deployment, scalability, and other issues that arise when real-world systems are used to process and compute data.

5 Edge inference

Edge training is achieved through the collaboration between multiple participants. However, edge inference can be performed not only through collaboration between cloud, edge and devices, but also through the model on a single device. In this section, we focus on the deployment and optimization of AI models on edge devices.

5.1 Inference architecture

We divide these inference architectures into three categories based on the inference participants: Solo-inference, Hybrid co-inference, and Peer-to-peer co-inference.

5.1.1 Solo-inference

Solo-inference refers to model inference being performed on a single working node. This working node is either device, edge or cloud. Therefore, Solo-inference can be further divided into three types: Device-only, Edge-only, and Cloud-only.

(1) Device-only: as shown in Fig. 5a, the inference task is performed entirely at the device. Obviously, this type of inference is only suitable for powerful devices, such as smartphones. In addition, it is necessary to optimize the model using model compression and inference acceleration for successful deployment on the device.

(2) Edge-only: as shown in the Fig. 5b, in this inference architecture, the model is deployed at the edge server. The device collects the data and sends it to the server, which performs inference and returns the results to the device. This type of inference has no performance requirements for the device. Moreover, it has lower latency than the Cloud-only inference architecture.

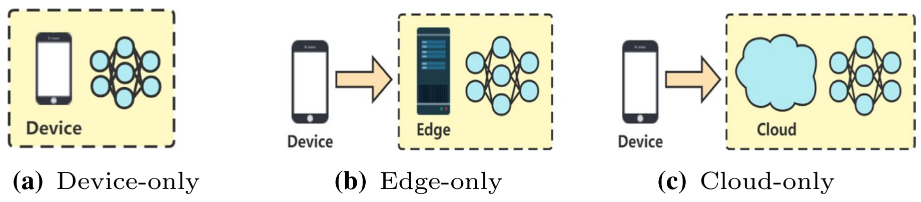


Fig. 5 Three types of Solo-inference

(3) Cloud-only: as shown in the Fig. 5c, in this inference architecture, the device collects data and uploads it to the cloud. Then the cloud processes the data and returns the results to the device. It is worth noting that most of existing DL services are based on this inference architecture. Although the cloud can provide powerful arithmetic support for inference tasks, this approach can cause communication bottlenecks and privacy leakage to some extent.

5.1.2 Hybrid co-inference

Hybrid co-inference allows inference tasks to be performed between the cloud, the edge, and devices. We divide these inference architectures into four categories based on the inference participants: Device-edge co-inference, Edge-cloud co-inference, Device-cloud co-inference and Device-edge-cloud co-inference.

(1) Device-edge co-inference: as shown in the Fig. 6a, in this inference architecture, the device first performs a portion of the inference task and then sends the results to the edge server for further inference. This approach can efficiently utilize the resources of edge devices.
(2) Edge-cloud co-inference: as shown in the Fig. 6b, in this inference architecture, the device only collects data and sends it to the edge server. Then, the edge server performs the initial inference and sends the results to the cloud to perform the remaining inference. Compared to the Cloud-only architecture, this approach alleviates the communication bottlenecks to some extent.
(3) Device-cloud co-inference: as shown in the Fig. 6c, in this inference architecture, the device and the cloud perform the inference task together. Compared with Cloud-only architecture, it is more beneficial to protect user privacy.
(4) Device-edge-cloud co-inference: as shown in the Fig. 6d, in this inference architecture, the computation resources of cloud, edge and devices are all effectively utilized. In addition, the user's privacy is protected to a certain extent.

5.1.3 Peer-to-peer co-inference

Different from Hybrid co-inference, the participants in Peer-to-peer co-inference are all peer nodes. We divide these inference architectures into two categories based on the difference of the participants: Co-inference between devices and Co-inference between processors.

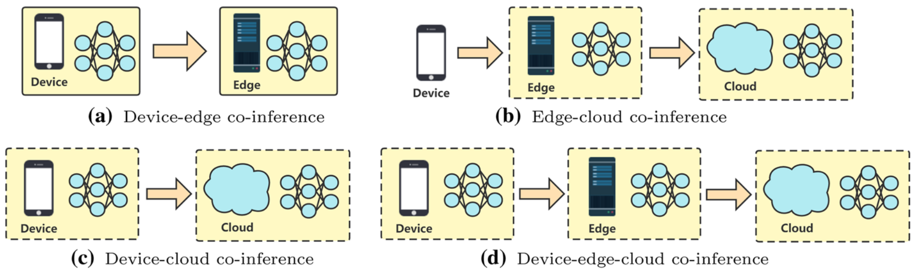

Device
Edge
(a) Device-edge co-inference
Device
Edge
Cloud
(b) Edge-cloud co-inference
Device
Cloud
(c) Device-cloud co-inference
Device
Edge
Cloud
(d) Device-edge-cloud co-inference

Fig. 6 Four types of Hybrid co-inference

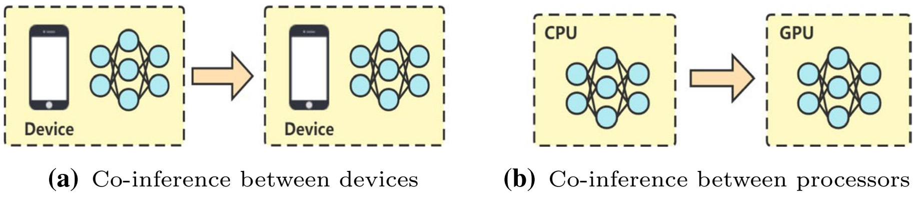

Device
Device
(a) Co-inference between devices
CPU
GPU
(b) Co-inference between processors

Fig. 7 Two types of Peer-to-peer co-inference

(1) Co-inference between devices: as shown in Fig. 7a, this inference architecture discards the edge and cloud, which cause computation burden on devices. However, different from Device-only architecture, it is not necessary for the devices in this architecture to be powerful, as collaboration between multiple devices can speed up convergence.
(2) Co-inference between processors: as shown in Fig. 7b, this inference architecture splits the inference task among different processors in the same device, thus reducing the energy consumption caused by long-term use of the GPU.

5.2 Model compression

The improvement of model performance is typically accompanied by the increasing of its depth or width. In the training phase, complicated DNN can be well-trained with the help of data centers. However, it is difficult to deploy such DNN to edge devices. The reason for this comes from two main sources. On the one hand, the model occupies too much storage. On the other hand, model inference consumes too much computation resources. Much of the existing efforts used model compression to reduce storage and accelerate inference. The efforts in this area mainly fall into five categories: model pruning, low-rank approximation, parameter quantization, knowledge distillation, and lightweight model design. We summarize the main efforts of model compression in Table 4.

5.2.1 Model pruning

Denil et al. (2013) demonstrated that most of the parameters in neural networks are redundant. They trained several networks by learning a few weights and predicting the remaining

Table 4 The main efforts on model compression

Related work	Method	Subdivision	Highlight
Han et al. (2015b)	Model pruning	Unstructured pruning	Threshold-based weight pruning
Guo et al. (2016)	Model pruning	Unstructured pruning	Recovering incorrect connection pruning
Molchanov et al. (2016)	Model pruning	Unstructured pruning	Taylor expansion-based weight pruning
Yang et al. (2017)	Model pruning	Unstructured pruning	Energy-aware pruning
Li et al. (2018a)	Model pruning	Lay-level pruning	Merging the tensor layers and non-tensor layers
Li et al. (2016b)	Model pruning	Filter-level pruning	Threshold-based filter pruning
Luo et al. (2017)	Model pruning	Filter-level pruning	Feature reconstruction error-based filter pruning
Yu et al. (2018)	Model pruning	Filter-level pruning	Evaluating the importance of neurons globally
You et al. (2019)	Model pruning	Filter-level pruning	Taylor expansion-based pruning globally
Luo and Wu (2020)	Model pruning	Filter-level pruning	Two-step pruning
Zuo et al. (2020)	Model pruning	Filter-level pruning	Pruning filters without damaging networks capacity
Lin et al. (2018a)	Model pruning	Kernel-level pruning	Synaptic strength-based pruning
Li et al. (2019d)	Model pruning	Kernel-level pruning	Kernel-sparsity and entropy-based pruning
Zhu et al. (2021)	Model pruning	Kernel-level pruning	Pruning kernels in both filters and layers
Denton et al. (2014)	Low-rank approximation	rank-k approximation	–
Jaderberg et al. (2014)	Low-rank approximation	Rank-1 approximation	–
Lebedev et al. (2014)	Low-rank approximation	CP decomposition	–
Kim et al. (2015)	Low-rank approximation	Tucker decomposition	Compressing and accelerating deep CNN
Lin et al. (2018b)	Low-rank approximation	–	Acting on convolutional and fully connected layers
Swaminathan et al. (2020)	Low-rank approximation	–	Introducing network pruning
Lee et al. (2021)	Low-rank approximation	–	Training low-rank CNN by alternatively compose-decompose
Gong et al. (2014)	Parameter quantization	Vector quantization	Clustering weights with K-means
Chen et al. (2015b)	Parameter quantization	Vector quantization	Clustering weights with a hash function
Wu et al. (2016)	Parameter quantization	Vector quantization	Accelerating and compressing CNN simultaneously
Courbariaux et al. (2015)	Parameter quantization	Binary quantization	Limiting the weights to two values
Courbariaux et al. (2016)	Parameter quantization	Binary quantization	Quantifying both activations and weights
Zhou et al. (2016)	Parameter quantization	Binary quantization	Quantifying activations, weights and gradients

Table 4 (continued)

Related work	Method	Subdivision	Highlight
Li et al. (2016a)	Parameter quantization	Ternary quantization	Limiting the weights to three values
Zhou et al. (2017)	Parameter quantization	Maintaining accuracy	Achieving five-bit lossless quantization
Mishra et al. (2017)	Parameter quantization	Maintaining accuracy	Increasing the width of the network for recovering accuracy
Lin et al. (2017a)	Parameter quantization	Maintaining accuracy	Using multiple binary weights to approximate the full precision weights
Kim et al. (2018)	Parameter quantization	Maintaining accuracy	Applying knowledge distillation for recovering accuracy
Rastegari et al. (2016)	Parameter quantization	Maintaining accuracy	Introducing a real-valued scaling factor for maintaining accuracy
Martinez et al. (2020)	Parameter quantization	Maintaining accuracy	Using real-valued activations to compute scaling factor
Hinton et al. (2015)	Knowledge distillation	Two-paticipant distillation	Instructing student with soft and hard labels
Romero et al. (2014)	Knowledge distillation	Two-paticipant distillation	Instructing student with intermediate representations
Zagoruyko and Komodakis (2016a)	Knowledge distillation	Two-paticipant distillation	Instructing student with attention graph
Kim et al. (2018)	Knowledge distillation	Two-paticipant distillation	Instructing student with interpreted information
Heo et al. (2019)	Knowledge distillation	Two-paticipant distillation	Instructing student with decision boundary
Zhou et al. (2018)	Knowledge distillation	Two-paticipant distillation	Training the teacher and student jointly
Zhang et al. (2018c)	Knowledge distillation	Two-paticipant distillation	Improve performance by mutual learning
Zhang et al. (2019)	Knowledge distillation	Self-distillation	Instructing shallow layers with deep layers
Mirzadeh et al. (2020)	Knowledge distillation	Three-paticipant distillation	Introducing a teaching assistant network
Lin et al. (2013)	Lightweight model design	–	Suggesting using 1×1 convolution for reducing FLOPs
Howard et al. (2017)	Lightweight model design	–	Introducing depth-wise separable convolution
Krizhevsky et al. (2012)	–	–	Introducing group convolution
Zhang et al. (2018b)	Lightweight model design	–	Group convolution and channel shuffle
Ma et al. (2018)	Lightweight model design	–	Proposing four guidelines for lightweight model design

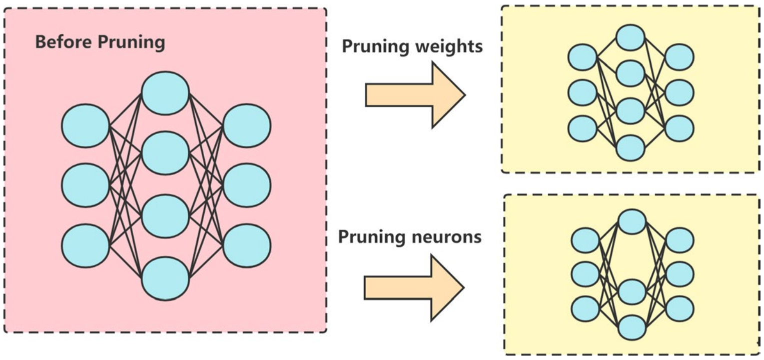


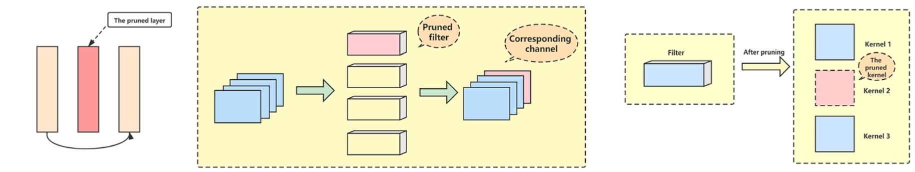
Fig. 8 Unstructured pruning

ones. The main idea of model pruning is to transform a dense network into a sparse network by removing redundant parameters from the network. According to the type of pruning, the existing efforts can be divided into unstructured pruning and structured pruning. Unstructured pruning can achieve a high compression rate, but it causes irregular memory access. In contrast, structured pruning can avoid this problem. However, it cannot achieve the same compression rate as unstructured pruning. Existing studies focus on structured pruning.

Unstructured pruning The earliest work on unstructured pruning can be traced back to LeCun et al. (1990). They used the Hessian matrix to represent the contribution of parameters and then removed parameters with less contribution. They demonstrated that this pruning method is more accurate than size-based pruning. To simplify the computation, this work assumed the Hessian matrix as a diagonal matrix, which can lead to incorrect weight removal. Different from LeCun et al. (1990), Hassibi et al. (1993) did not impose any restrictions on the Hessian matrix and achieved better effects. However, these methods are mainly based on second-order derivatives, which can cause expensive computation costs. Therefore, they are not suitable for deep models.

As shown in Fig. 8, Han et al. (2015b) proposed a threshold-based weight pruning method. First, the weights of the connections are obtained by training the network. Then connections whose weights are below a certain threshold are removed. Finally, the pruned network is retrained to fine-tune the weights of the remaining connections. This approach achieved 9× and 13× pruning rates for AlexNet and VGG-16, respectively, while maintaining accuracy. It is highly effective for network compression. However, it has several drawbacks: (1) it consumes a lot of time and computing resources; (2) the authors assumed that connections with lower weights contribute less, which may lead to important connections being removed; (3) it leads to irregular memory access, so it only compresses the model and does not speed up the inference; (4) they set a threshold for all layers, which may not be optimal for a single layer.

Several works improved the work of Han et al. (2015b) from different perspectives. Guo et al. (2016) proposed dynamic network surgery. This method contains two operations, pruning and splicing. Among them, the splicing is to prevent important connections from being removed. Molchanov et al. (2016) used a new criterion based on Taylor expansion to measure the importance of connections and showed better performance than other criterias. Han et al. (2016) proposed an efficient inference engine (EIE) to accelerate the



(a) Layer-level pruning **(b)** Filter-level pruning **(c)** Kernel-level pruning

Fig. 9 Three levels of pruning

inference of compressed models. Manessi et al. (2018) jointly optimized thresholds and network weights to achieve a higher compression rate. Moreover, since each layer has a threshold, a single layer can be optimized independently of the other layers. Building on Han et al. (2015b), Han et al. (2015a) further compressed DNN with quantization and huffman coding. The storage required for AlexNet and VGG-16 was reduced by a factor of 35 and 49, respectively. However, huffman coding can cause inconvenience to the inference phase because its decoding process brings additional computation overhead.

The methods mentioned above reduced the model size by pruning unimportant parameters of the model, but the energy consumption in the inference phase was not considered. Yang et al. (2017) argued that although the model parameters are mainly derived from the fully connected layers, the convolutional layers account for most of the energy consumption. Thus, the amount of weights does not serve as a direct indicator of energy consumption. Therefore, they proposed an energy-aware pruning approach. It can identify the most energy-consuming part of the CNN for energy-saving.

Structured pruning The model obtained after structured pruning remains the same in terms of network structure. As a result, it can be perfectly supported by existing DL libraries. According to the pruning level, the existing efforts can be divided into four categories: layer-level pruning, filter-level pruning, kernel-level pruning and intra-kernel pruning. It is worth noting that filter-level pruning and channel pruning are the same concept. The reason for this is that when a filter is pruned, its corresponding channel is also trimmed (as shown in Fig. 9b). Since intra-kernel pruning has been less studied, here we mainly present the first three pruning efforts.

(1) Layer-level pruning Huang et al. (2016) proposed stochastic depth. During training, they randomly skipped some layers to shorten the network. The network with deeper layers was used for testing. This simple and effective method significantly reduces the training time and testing error. Wen et al. (2016) proposed a structural sparse learning (SSL) method to regularize the structure of DNN. Specifically, during training, SSL learns a compact structure directly through Group Lasso, which can reduce the number of layers in the network. Li et al. (2018a) proposed DeepRebirth. The authors found that layers with few parameters consume a lot of computation time due to memory access speed and other factors. They divided the network layers into tensor and non-tensor layers. By merging the tensor and non-tensor layers, the run-time can be significantly decreased. Layer-level pruning does not cause irregular memory access. However, the whole layer is pruned, so layer-level pruning is not flexible. In contrast, filter-level pruning has great flexibility and ease-of-implementation (Liu et al. 2017).
(2) Filter-level pruning The key of filter-level pruning is to measure the importance of filters. Li et al. (2016b) proposed filter-level pruning. They used the absolute value of weights to measure the importance of filters. Liu et al. (2017) used the scaling factor

of the BN layer to evaluate the importance of filters. Specifically, they performed L1 regularization on the scaling factor and removed channels with a scaling factor of 0 after regularization. Luo et al. (2017) presented ThiNet, an efficient and uniform CNN framework. It crops channels by minimizing feature reconstruction errors. Similarly, (He et al. 2017) also performed channel selection by minimizing the feature reconstruction error. The difference is that the former used a greedy algorithm for channel selection, while the latter used Lasso regression for channel selection. Luo et al. (2017) and He et al. (2017) only considered the importance of neurons in a single layer or two consecutive layers. However, trimming neurons that seem to be unimportant may have an influence on the response output of subsequent layers. Therefore, the importance of neurons needs to be considered in a holistic manner. Yu et al. (2018) demonstrated that greedy layer-by-layer pruning leads to severe reconstruction error propagation. In addition, they proposed the Neuronal Importance Score Propagation (NISP). Specifically, they defined the second-to-last layer before classification as the final response layer (FRL) and decided the importance of each feature in the FRL by feature selection. Then, the importance scores were back-propagated to the neurons in all layers. Finally, unimportant neurons were cropped. You et al. (2019) estimated the importance of the global filters by Taylor expansion. Specifically, they multiplied the output of the CNN module with the channel scaling factor. When the channel scaling factor is set to 0, the corresponding filter is removed. Taylor expansion was used to estimate the change in the loss function and rank the importance of the global filters. Finally, unimportant filters are removed. All the efforts mentioned above treated pruning and fine-tuning as two independent steps. In contrast, (Luo and Wu 2020) combined the two steps and proposed an end-to-end training system, AutoPruner. They proved that the two steps can be mutually beneficial. Specifically, they took the activation of the previous layer as input, which resulted in a binary code. Then, filters corresponding to the value of 0 in the binary code were removed. Zuo et al. (2020) found that previous filter-level pruning corrupted the network capacity. To cope with this problem, they proposed a filter-level pruning method with no damage to the network capacity. Specifically, they pruned redundant filters in the network. Then, new feature maps are generated based on the remaining feature maps to recover the original capabilities.

(3) Kernel-level pruning Filter-level pruning allows pruning the filters and their corresponding channels for compressing and accelerating the CNN. However, it causes dimensionality mismatch in multi-branch networks. As shown in Fig. 10, in which n denotes the number of filters that are pruned. In addition, the kernels in the filter may still have an important role even if the filter is regarded as redundant. Therefore, filter-level pruning may result in loss of accuracy of the model (Zhu et al. 2021). This was also demonstrated by both (Anwar et al. 2017; Zuo et al. 2020). Anwar and Sung (2016) proposed kernel-level pruning for reducing model complexity. Anwar et al. (2017) explored filter-level, kernel-level, and intra-kernel pruning. They demonstrated that the expressive ability of the model would be impaired when filter-level pruning is applied in a high ratio. Lin et al. (2018a) used synaptic strength to measure the importance of connections. The non-important connections are identified by sparse regularization. Then connections with lower synaptic strength are pruned to generate kernel-level sparse CNN. Li et al. (2019d) combined kernel-sparsity with entropy to measure the importance of feature maps. In addition, they utilized kernel clustering to reduce the number of kernels. Zhu et al. (2021) designed a progressive kernel-level pruning. Specifically, they pruned kernels in both filters and layers, which can avoid the local optimum that occurs when pruning directly in the layers.

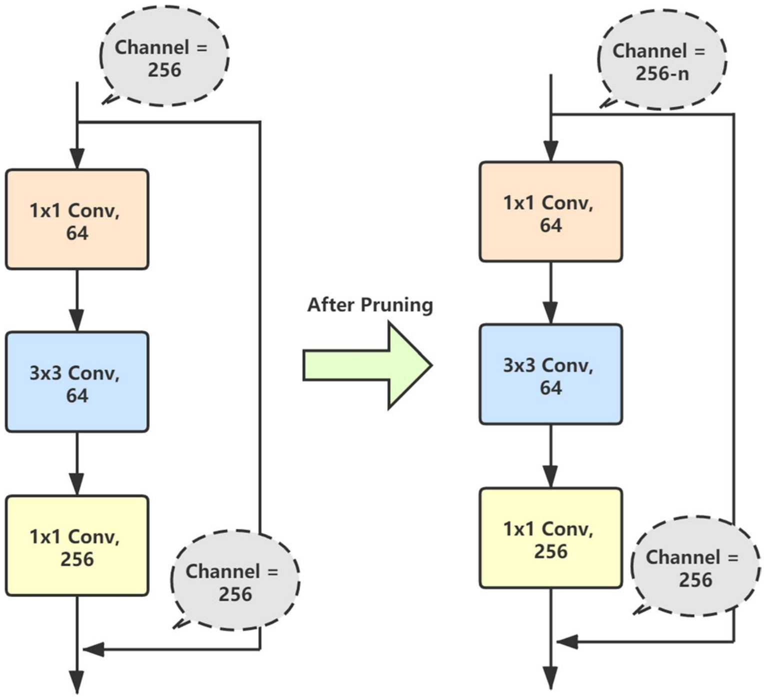


Fig. 10 The illustration of channel mismatch problem

5.2.2 Low-rank approximation

Decomposing a matrix into two smaller matrices can improve computation efficiency. For example, a $m \times k$-dimension weight matrix W can be decomposed into $m \times d$ and $d \times k$ -dimension matrices. The computation complexity of the matrix W is $O(m \times k)$, while the computation complexity of the decomposed matrix is $O(m \times d + d \times k)$. The computation can be effectively decreased by controlling the value of d. Tensor can be decomposed if we extend this idea to multi-dimensional space. The existing decomposition methods can be divided into four categories: Tensor Train decomposition, Singular Value Decomposition (SVD), Canonical Polyadic (CP) Decomposition and Tucker decomposition.

A part of the early efforts focused on the decomposition of general DNN. For example, to reduce the parameters in DNN, Sainath et al. (2013) decomposed the final weight layer of DNN with low-rank matrix. Similarly, Xue et al. (2013) used SVD to decompose the weight matrices in DNN for reducing the model size while maintaining accuracy.

With the development of CNN, this method has been widely applied in CNN. Rigamonti et al. (2013) first proposed to accelerate convolution operations with low-rank approximation. Denton et al. (2014) used low-rank approximation and filter clustering to achieve the 2-3x speed-up in the first convolutional layer. Similarly, Jaderberg et al. (2014) decomposed the $w \times h$-dimension convolutional layer matrix of into $w \times 1$-dimension matrix and $1 \times h$-dimension matrix. Lebedev et al. (2014) used CP decomposition to decompose a convolutional layer into multiple layers of low complexity. In contrast to the methods mentioned above, Novikov et al. (2015) mainly targeted the fully-connected layer. They converted the dense weight matrix in the fully-connected layer to the Tensor Train format. The expressive power of the original network is preserved while reducing the parameters.

Denton et al. (2014), Jaderberg et al. (2014) and Lebedev et al. (2014) only focused on the decomposition of one or several layers. These methods show good acceleration on shallow models, but lack acceleration on deep models. Zhang et al. (2015) proposed a method to compress and accelerate deep CNN. They considered non-linear units and reduced the error accumulation when using low-rank approximation for multi-layer CNN. This method achieved a 4x full-model speed-up with only a 0.3% increase in top-5 error for the ImageNet classification task. Kim et al. (2015) achieved acceleration of the entire model with Tucker decomposition. First, they used Variational Bayesian Matrix Factorization for rank selection. Then Tucker decomposition was performed on the kernel tensors. Finally, the entire model was fine-tuned. Block term decomposition (BTD) based on low-rank and group sparse was proposed in Wang and Cheng (2016). For convolutional layers, each kernel tensor is decomposed into the sum of a few low-rank tensors. These low-rank tensors are used to replace the original tensor in the convolutional layer. Astrid and Lee (2017) successfully applied CP decomposition for full model compression. Due to the instability of CP decomposition, previous methods (Lebedev et al. 2014) did not successfully compress the entire CNN with CP decomposition. The authors argued that CP decomposition of the entire CNN would lead to error accumulation. They overcame the instability of CP decomposition by iterative fine-tuning.

Most of the aforementioned efforts focused on accelerating the convolutional layers (Denton et al. 2014; Jaderberg et al. 2014; Lebedev et al. 2014) or compressing the fully connected layers (Novikov et al. 2015) instead of aiming for joint optimization. Lin et al. (2018b) proposed a holistic CNN compression framework which acts on both the convolutional and fully connected layers. Specifically, they presented a global CNN compression method based on low-rank decomposition and knowledge transfer. They used a low-rank decomposition with a closed-form solver to accelerate the convolution computation and reduce the memory overhead.

To further compress the network, Swaminathan et al. (2020) introduced pruning techniques. They considered the importance of neurons in the low-rank decomposition and achieved better compression rate by keeping the unimportant neurons at low-rank. The above decomposition methods were trained in a decomposition manner. To further accelerate inference, Lee et al. (2021) proposed an alternating tensor compose-decompose (ATCD) method for training low-rank CNN. For example, for a rank-1 CNN, in the training phase, the 3-D filters of rank-1 CNN are decomposed into one-dimension vectors to form 3-D filters again. In the test phase, the 3-D filters are permanently decomposed into one-dimension vectors. This allows the rank-1 CNN to maintain its original accuracy, but with the same inference speed as the one-dimension vector.

5.2.3 Parameter quantization

Parameter quantization is an effective method for network compression and acceleration and is widely used in the field of image compression. According to the nature of these works, we divide these efforts into two main parts: vector quantization and low-bit quantization.

Vector quantization The main idea of vector quantization is to divide the weights into groups and each group shares the same weight. Gong et al. (2014) used vector quantization for network compression. They focused on operating on the fully connected layers. Therefore, this work does not involve inference acceleration. The authors found that 8-16 compression ratio can be achieved by simply using the K-means algorithm for quantization with no more than 0.5% loss in top-5 accuracy. Chen et al. (2015b) proposed HashNets. First, they used a hash function to classify the connection weights. Then, connections that were grouped into the same hash bucket shared a common weight. Chen et al. (2016) further extended this idea. They transformed the filter weights to the frequency domain. Then a hash function was used to group the parameters in the frequency domain.

The efforts mentioned above compressed the model without accelerating the inference. Wu et al. (2016) proposed a product quantization-based approach to simultaneously accelerate and compress CNN. They quantified both convolutional layers and fully connected layers by minimizing the response error.

Low-bit quantization In general, the parameters of neural networks are represented by 32-bit floating-point numbers. In fact, it is not necessary to maintain such a high accuracy. Quantification can reduce the storage at the expense of accuracy. Therefore, the model size can be reduced while maintaining accuracy by rationally quantizing the network.

In the inference phase, floating-point multiplication consumes a lot of time and computation resources, while fixed-point quantization can effectively alleviate this problem. There are two main reasons for this. First, fixed-point computation is typically faster than floating-point and consumes much less energy and hardware resources. In addition, lower precision data representation reduces memory footprint, allowing larger models to be deployed on resource-constrained devices.

Holi and Hwang (1993) first provided a theoretical analysis of the error caused by the finite precision computation. They demonstrated that 8–16 bits of quantization is sufficient to train a small neural network. However, Chen et al. (2014) found that although 16-bit fixed point format can meet the need for inference to a great extent, it may lead to a decrease in accuracy or even prevent the model from converging. In a further step, Gupta et al. (2015) presented stochastic rounding, which allows training DNN using 16-bit fixed point with almost no degradation in classification accuracy.

Binary quantization, as a special case of low-bit quantization, can theoretically achieve 32× compression rate. By limiting the weights to two values (e.g., 1 or −1), the multiplication operations can be replaced with additive operations. Since the multiplier is the most computationally resource-intensive component, binarization operation can bring significant benefits to the hardware used for DL (Courbariaux et al. 2015). Soudry et al. (2014) proposed the expectation back propagation (EBP) algorithm. They binarized the network by Variational Bayesian and proved the high performance of the binary network trained with EBP. Courbariaux et al. (2015) presented BinaryConnect. They constrained the weights to 1 or −1, which eliminates the multiplication operation in the forward propagation and thus accelerates the training process. However, experiments in Rastegari et al. (2016) showed that this method cannot be applied to large-scale datasets.

If only the weights are quantified, time-consuming arithmetic operations are still required. Courbariaux et al. (2016) further extended the work of Courbariaux et al. (2015). They quantified both activations and weights. In this case, bit-wise operations can be used in forward-propagation, thus further accelerating the inference. A similar approach was adopted by Rastegari et al. (2016). In Courbariaux et al. (2016) and Rastegari et al. (2016), although the weights are binary, the gradients are full precision. Therefore, bit-wise operations cannot be used in the back-propagation. This means that during the training process, Courbariaux et al. (2016) and Rastegari et al. (2016) would spend a lot of time in the back-propagation. To further speed up the inference, the gradients were also quantified in Zhou et al. (2016). Specifically, the gradients were randomly quantified to low-bit during back-propagation. Since the weights, activations, and gradients during forward-propagation and back-propagation are all low-bit, bit-wise operations can be used during the whole training process.

Although the network after low-bit quantization can lead to high compression rate, this method leads to a decrease in accuracy. For example, the test results of binarized neural network (Courbariaux et al. 2016) on the ImageNet dataset showed that the difference in top-1 accuracy between real-valued ResNet-18 and binary ResNet-18 is a staggering 28%. Therefore, many existing works have focused on how to maintain model accuracy after low-bit quantization. Li et al. (2016a) proposed ternary weight networks (TWN). They restricted the weights to 1, 0, or -1 thus achieving a trade-off between accuracy and compression ratio. Zhou et al. (2017) presented incremental network quantization (INQ), which can convert a pre-trained full-precision network into a lossless low-precision network. INQ contains three processes: weight partition, group-wise quantization, and retraining. And it can achieve five-bit lossless quantization. Mishra et al. (2017) compensated the accuracy degradation caused by low-bit quantization through increasing the width of the network. Lin et al. (2017a) proposed ABC-Net. They used a linear combination of multiple binary weights to approximate full precision weights. Kim et al. (2018) combined low-bit quantization and knowledge distillation. They proved that knowledge distillation can significantly improve the accuracy of low-bit networks. Hu et al. (2018b) presented a hash-based training method for binary networks. They revealed the tight connection between inner product hashing and binary networks. In addition, the authors demonstrated that the network parameter binarization problem can be converted into a hash problem. This method can effectively reduce the accuracy loss.

Rastegari et al. (2016) proposed the binary weight network (BWN) and XNOR-Net. They introduced a real-valued scaling factor to adjust the output of the binary convolution, which can compensate the error caused by the binary quantization to some extent. On the foundation of XNOR-Net, Bulat and Tzimiropoulos (2019) proposed XNOR-Net++. They fused the activation and weight scale factor into a single factor, which is learned by back-propagation. Compared with XNOR-Net, the accuracy of XNOR-Net++ is improved by 6%. Similar to Bulat and Tzimiropoulos (2019), Rastegari et al. (2016) and Martinez et al. (2020) improved the performance of the network by rescaling the output of the binary convolution. Specifically, before binarization, they used the real-valued activation of the binary network to compute a scaling factor. After applying the binary convolution, this scaling factor was utilized to adjust the activations. Compared to XNORNet++, this method improves 8.3% accuracy on Top-1 error.

5.2.4 Knowledge distillation

The core idea of knowledge distillation is to train a lightweight neural network with the help of a large neural network. The large neural network is called the teacher network, and the lightweight neural network is called the student network. Knowledge distillation is a learning process for the student network. What knowledge to learn and how to learn the knowledge are two cores of knowledge distillation. According to the number of

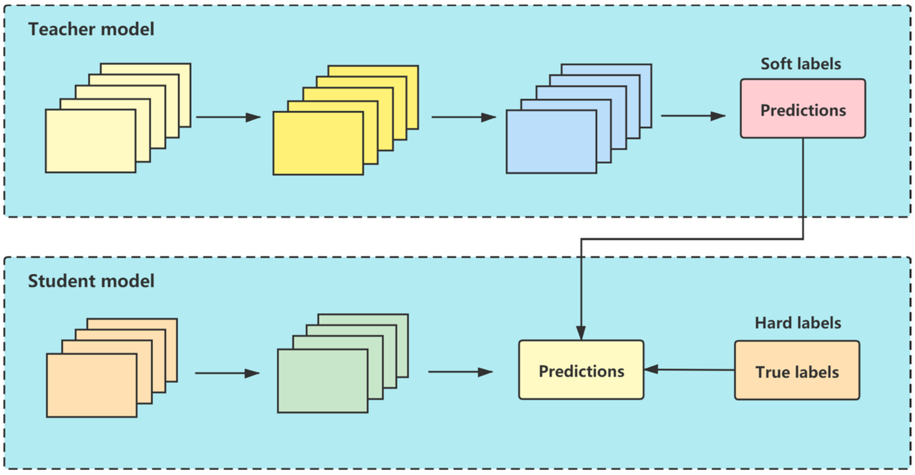


Fig. 11 Knowledge distillation

participants in the knowledge distillation, we classify these efforts into three categories: three-paticipant distillation, two-paticipant distillation and self-distillation.

Long before the concept of knowledge distillation was introduced, Buciluǎ et al. (2006) attempted to have a small model learn the representation of a large model. Specifically, they used a data generation algorithm to obtain unlabeled data and labeled them with a well-trained teacher network. These data contain the knowledge of the teacher network. The student network trained with these data performs better than the one trained with original data. However, this approach is not suitable for deep models. Ba and Caruana (2013) extended this work to a shallow and wide student network. In which, the output of the teacher network was used to train the student network. The trained student network has similar performance to the teacher network. Hinton et al. (2015) first proposed knowledge distillation. As shown in Fig. 11, they obtained soft labels utilizing a well-trained teacher network. Then, both soft and true labels are used to train a small network. Sau and Balasubramanian (2016) argued that the knowledge from a single teacher may be restricted. Building on this idea, they proposed a noise-based regularization method to mimic the guidance of multiple teachers.

Besides mimicking the output of teachers, some efforts suggest learning the hidden features of the teacher network. For example, Romero et al. (2014) presented FitNets. They used intermediate representations of the teacher network to train the student network. Yim et al. (2017) argued that simply learning the intermediate representations of the teacher network may constrain the performance of the student network. Therefore, they defined distillation knowledge as a feature relationship between layers and utilized the flow of solution procedure (FSP) matrix to represent it. Zagoruyko and Komodakis (2016a) introduced attention mechanisms to knowledge distillation. The attention map of the teacher network is utilized to train the student network, which significantly improved the performance of the student network. Kim et al. (2018) suggested learning the interpreted information of the teacher network. They used two convolutional modules as paraphraser and translator. The paraphraser extracted the features of the teacher network as teacher factors and the translator extracted the features of the student network as student factors. The translator of the student network can help the student understand the teacher factors of the teacher network.

Considering the relevance of the generalization performance of a classifier to its decision boundary, Heo et al. (2019) defined distillation knowledge as the decision boundary. Specifically, they used self-adversarial techniques to move the original samples near the decision boundary. The moved samples are called boundary support samples (BSS), which contain information about the decision boundary. Training the student network with BSS allows the student network to better learn the decision boundary of the teacher network.

All the efforts mentioned above required two steps to obtain a lightweight student model: (1) training a complex teacher network. (2) using the teacher network to train the student network. This process is time-consuming. Besides that, it is difficult to find a suitable teacher. For the former, several efforts proposed to train the teacher network and the student network together. For example, Mishra and Marr (2017) presented a scheme to jointly train the teacher network and the student network. Similarly, Zhou et al. (2018) argued that the knowledge of the teacher network exists not only in the final output, but also in the training process of the entire network. Based on this idea, they presented Rocket Launcing, which consists of a light net and a booster net. Specifically, in the training phase, the light net and the booster net are trained on the same task. Meanwhile, the light net continuously learns the knowledge of the booster net, thus improving the performance of the light net. Zhang et al. (2019) proposed self-distillation. Specifically, they added several branches to the network, which were trained with the help of the main network.

Although (Mishra and Marr 2017; Zhang et al. 2019; Zhou et al. 2018) improved the time-consuming process of knowledge distillation, they did not address the impact of the teacher-student performance gap on training student networks. To solve this problem, Mirzadeh et al. (2020) introduced a teaching assistant network. Their experiments showed that the performance of the student network would decrease if the gap between teacher and student is relatively large. To bridge the gap between the student network and the teacher network, they introduced the intermediate model (i.e., the teaching assistant network). Unlike the work of Mirzadeh et al. 2020, Zhang et al. 2018c improved network performance through mutual learning of student networks. This work found that a powerful teacher network is not essential and that the student network obtained by mutual learning performs better than the student network guided by the teacher network. With this approach, student networks can be trained well while eliminating the impact of performance gaps between networks on training.

5.2.5 Lightweight model design

Much of the existing works improved the performance of the network by increasing the width and depth of the network (Szegedy et al. 2015; He et al. 2016), which makes it difficult to deploy the model on resource-constrained devices. To enable the model to be deployed on edge devices, an increasing amount of works has focused on designing efficient and lightweight network model. 1×1 convolution, group convolution and depth-wise separable convolution have been widely used in lightweight model design because they can effectively reduce floating-point operations per second (FLOPs).

Lin et al. (2013) proposed the Network-In-Network (NIN). To reduce the computation complexity, they suggested using 1×1 convolution and replacing the fully connected layers with global average pooling layers. GoogleNet Szegedy et al. (2015) and ResNet He et al. (2016) both adopt these ideas. In addition, the earliest lightweight model, SqueezeNet Iandola et al. (2016), also made extensive use of 1×1 convolution instead of 3×3 convolution.

Howard et al. (2017) proposed depth-wise separable convolution and MobileNetV1. Depth-wise separable convolution is composed of depth-wise convolution and point-wise convolution. The depth-wise convolution convolves each channel of the input image separately, and the point-wise convolution uses 1×1 convolution kernels to convolve the feature maps generated by the depth-wise convolution. Compared with common convolution, depth-wise separable convolution reduces the parameters by a factor of 9. Guo et al. (2018a) analyzed the mathematical relationship between common convolution and depth-wise separable convolution. They proved theoretically that common convolution and depth-wise separable convolution are similar in effect. Likewise, Xception (Chollet 2017) also used depth-separable convolution. However, this work aimed at improving network performance rather than reducing parameters. On the basis of MobileNetV1, Sandler et al. (2018) proposed MobileNetV2. They introduced an inverted residual structure with a linear bottleneck. Compared with MobileNetV1, the accuracy is improved and the model size is further reduced. In a further step, Howard et al. (2019) proposed MobileNetV3. They improved blocks by updating the activation function and introducing the Squeeze-and-Excite module (Hu et al. 2018a). MobileNetV3-Large improves the accuracy by 3.2% over MobileNetV2 in the ImageNet classification task, while reducing the 20% latency.

Group convolution is a method between common convolution and depth-wise separable convolution. Given a group number n, it can reduce the number of parameters by a factor of n. The earliest use of group convolution can be traced back to AlexNet (Krizhevsky et al. 2012). Xie et al. (2017) also used group convolution and proposed ResNeXt. However, the aim of ResNeXt is to improve performance instead of reducing parameters. Although 1×1 convolution has been proven its ability to reduce model complexity by a large number of works (Szegedy et al. (2015), He et al. (2016), Iandola et al. (2016), Lin et al. (2013)), Zhang et al. (2018b) argued that dense 1×1 convolution can degrade the performance of the model. Based on this idea, they used group convolution to reduce the complexity of 1×1 convolution. In addition, the authors proposed channel shuffle to fuse the feature information of different groups. The works mentioned above mainly designed lightweight models by reducing FLOPs. However, Ma et al. (2018) pointed out that direct metrics (e.g., speed) rather than indirect metrics (e.g., FLOPs) should be considered when designing network architectures. Based on this, the authors proposed four guidelines for designing network architectures and a new network, ShuffleNetV2.

Previous works improved the performance of the model by increasing the depth (He et al. 2016) and width (Zagoruyko and Komodakis 2016b) of the network. Tan and Le (2019) investigated the effects of width, depth, and image resolution on the network and presented EfficientNet, where EfficientNet-B7 is 8.4× smaller and 6.1× faster compared to the model with the highest accuracy back then. However, the increase in image resolution leads to significant memory usage. To address this problem, Tan and Le (2021) proposed a progressive learning strategy that adjusts the regularization factor according to the image size.

Most of the networks mentioned above are designed by humans. In addition, there are some works that used Neural Network Architecture Search (NAS) to build networks. For example, EfficientNet-B0 in Tan and Le (2019) is designed through NAS. Since NAS consumes a lot of resources and these works are not oriented towards resource-constrained edge devices, we do not present it here. For readers who are interested in NAS, you can view (Elsken et al. 2019; Wistuba et al. 2019).

5.3 Inference acceleration

Due to the limited energy of edge devices, it is important to accelerate inference thus reducing latency and lowering energy consumption. According to the difference of these efforts, we divide them into three categories: model partition, edge caching, and conditional computation. Among them, the conditional computation can be further divided into model early exit, input filtering, and model selection. We summarize the works on inference acceleration in Table 5.

5.3.1 Model partition

The general DNN services are provided through the cloud, which can result in high bandwidth usage and privacy concerns. Kang et al. (2017b) evaluated DNN services on both cloud and edge devices. They found that uploading data to the cloud is the largest bottleneck for DNN services due to the huge data transfer load. Although edge computing can reduce transmission overhead, compute-intensive tasks are difficult to be deployed on resource-constrained devices. By partitioning DNN between cloud, edge and devices, the requirements of DNN services in terms of energy consumption, latency and throughput can be met. According to the difference of DNN partition, we classify these works into the following two categories: (1) hybrid partition and (2) peer-to-peer partition.

Hybrid partition As shown in Fig. 12, Hybrid partition consists of partition between device-edge, partition between device-cloud and partition between device-edge-cloud. A key point of these efforts is how to determine the optimal partition point. Kang et al. (2017b) proposed Neurosurgeon, which is capable of predicting the latency and energy consumption of each DNN layer without performing DNN. Based on these predictions, the current bandwidth and data center load level, an optimal DNN partition point that meets the energy consumption and latency requirements can be found. Ko et al. (2018) partitioned the DNN at the end of the convolutional layers. Moreover, they reduced the energy consumption of the edge and the bandwidth consumption during transmission by performing lossy encoding on the feature map. Finally, they retrained DNN on the host after DNN partition to mitigate the accuracy degradation caused by lossy coding. Similarly, Li et al. (2018b) also compressed the feature map before it is transferred to the cloud. Moreover, for different models and bandwidth situations, they found a correct splitting point using integer linear programming.

Besides finding the optimal DNN partition point, there are many challenges for DNN partition. For example, the instability of the network, the scheduling problem when multiple devices request DNN services, and the impact of user mobility on DNN services. Several works optimized model partition with these perspectives. Considering the real-world network environments, Li et al. (2019b) designed two strategies for static and dynamic environments. Moreover, to further improve the performance, they combined model partition with model-early-exit and proposed Edgent, which can maximize the inference accuracy while satisfying the latency requirement. DeePar (Huang et al. 2019) divided DNN between cloud, edge and devices. They considered not only the DNN partition optimization problem, but also the multi-task scheduling problem. To reduce the delay caused by the mobility of users to the DNN partition, Tian et al. (2021) proposed a mobility-included DNN partition offloading algorithm (MDPO). It is worth noting that MDPO is applicable not only to DNN with chain topology [e.g., AlexNet (Krizhevsky et al. 2012)], but also to DNN with directed acyclic graph topology [e.g., ResNet (He et al. 2016)].

Table 5 The main efforts on inference acceleration

Related work	Method	Algorithm/framework	Highlight
Kang et al. (2017b)	Device-cloud partition	Neurosurgeon	Predicting the latency and energy consumption of each DNN layer
Li et al. (2018b)	Device-cloud partition	JALAD	Compressing the output feature maps of the edge devices
Ko et al. (2018)	Device-edge partition	–	Reducing bandwidth consumption by lossy coding
Li et al. (2019b)	Device-edge partition	Edgent	Providing two strategies for static and dynamic environments
Tian et al. (2021)	Device-edge partition	MDPO	Considering the mobility of users
Huang et al. (2019)	Device-edge-cloud partition	Deepar	Considering multi-task scheduling problem
Mao et al. (2017a)	Partition between devices	MoDNN	Proposing two segmentation schemes for convolutional layers and fully connected layers
Mao et al. (2017b)	Partition between devices	MeDNN	Presenting greedy two dimensional Partition and structured model compact deployment
Zhou et al. (2019a)	Partition between devices	AAIOT	Considering computation and communication capabilities of devices
Zhao et al. (2018b)	Partition between devices	DeepThings	Proposing fusion tile partitioning
Stahl et al. (2021)	Partition between devices	DeeperThings	Considering weight partitioning
Lane et al. (2016)	Partition between processors	DeepX	Proposing runtime layer compression (RLC) and deep architecture decomposition (DAD)
Chen et al. (2015a)	Data cache	Glimpse	–
Liu et al. (2019)	Data cache	–	Designing dynamic ROI encoding and parallel stream and inference for reducing latency
Apicharttrisorn et al. (2019)	Data cache	MARLIN	No need to offload tasks to the cloud or edge
Liu et al. (2020)	Data cache	AdaVP	Running target detector and target tracker in parallel
Drolia et al. (2017b)	Computation results cache	Cachier	No need to reasoning sample again
Drolia et al. (2017a)	Computation results cache	Precog	Storing the computation results to the device in advance
Teerapittayanon et al. (2016)	Model early exit	Branchynet	Using the entropy of the classification result as judging criteria
Panda et al. (2017)	Model early exit	Conditional deep learning	Using the maximum confidence in the output of each stage as judging criteria
Laskaridis et al. (2020)	Model early exit	SPINN	Combining model early exit with model partition
Lo et al. (2017)	Model early exit	–	Defining thresholds using statistical methods
Bolukbasi et al. (2017)	Model early exit	–	Introducing a decision function as judging criteria
Li et al. (2018c)	Model early exit	DeepIns	–

Table 5 (continued)

Related work	Method	Algorithm/framework	Highlight
Zeng et al. (2019)	Model early exit	Boomerang	Finding optimal partition points and exit points using deep reinforcement learning
Lee et al. (2020)	Model early exit	–	Evaluating the effect of the number of exit points on inference performance
Kang et al. (2017a)	Filtering of video data	Noscope	Performing inference only in the specified task
Hsieh et al. (2018)	Filtering of video data	Focus	A two-stage filtering system
Zhang et al. (2020)	Filtering of video data	FFS-VA	A three-stage filtering system
Jain et al. (2020)	Filtering of video data	ReXCam	A cross-camera filtering system
Wang et al. (2018a)	Filtering in edge offloading	–	–
Canel et al. (2019)	Filtering in edge offloading	FilterForward	Optimizing the multi-tenant problem
Pakha et al. (2018)	Filtering in edge offloading	–	Allowing the server to decide which frames to filter
Park et al. (2015)	Selection of independent model	BL-DNN	Selecting appropriate model according to the difficulty of inference
Stamoulis et al. (2019)	Selection of independent model	–	Treating the model selection as a hyperparametric optimization problem
Taylor et al. (2018)	Selection of independent model	–	Selecting the optimal DNN according to the desired accuracy and inference time dynamically
Yang et al. (2020b)	Selection of independent model	RANet	–
Tann et al. (2016)	Selection of partial model	–	No need to store the weights of multiple networks
Panda et al. (2016)	Selection of partial model	FALCON	Constructing a tree of classifiers
Wang et al. (2018b)	Selection of partial model	SkipNet	–

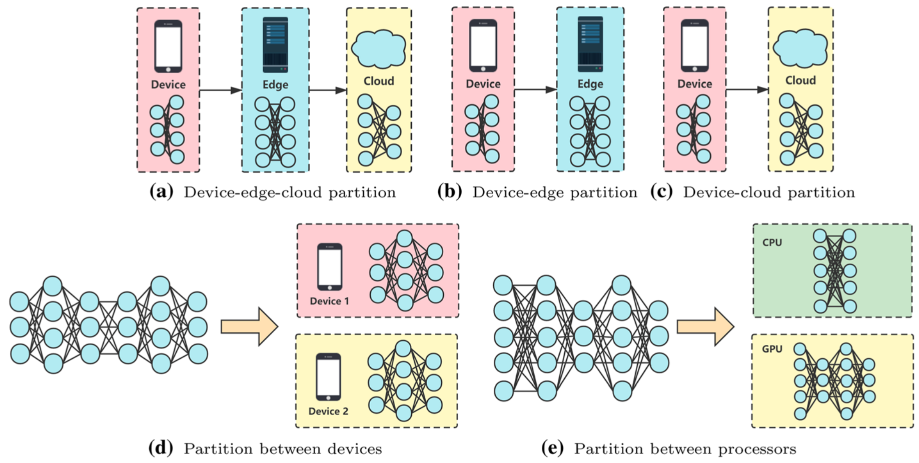

Device
Edge
Cloud
(a) Device-edge-cloud partition
Device
Edge
(b) Device-edge partition
Device
Cloud
(c) Device-cloud partition
Device 1
Device 2
(d) Partition between devices
CPU
GPU
(e) Partition between processors

Fig. 12 Five types of DNN partition

Peer-to-peer partition As shown in Fig. 12, Peer-to-peer partition consists of partition between devices and between processors. It discards the edge and cloud, partitioning DNN between devices or between processors. For example, Mao et al. (2017a) partitioned the trained DNN model onto multiple devices to accelerate inference. In the computing cluster, the device carrying the test data is the group owner and the other devices act as working nodes. Moreover, they proposed two partition schemes according to the difference of convolutional layers and fully-connected layers. Mao et al. (2017b) proposed greedy two dimensional Partition (GTDP), which can adaptively partition a DNN to different devices according to the resources of mobile devices. In addition, they presented structured model compact deployment (SMCD), which utilizes structured sparse pruning to accelerate DNN execution. Zhou et al. (2019a) traded off the computation and communication capabilities of devices and designed a dynamic programming algorithm to find the optimal partition policy under the criterion of minimizing the system response time.

Zhao et al. (2018b) proposed fusion tile partitioning (FTP) to divide CNN into independent tasks. Additionally, they considered the allocation of FTP in dynamic scenarios and presented a new scheduling scheme to maximize the reuse of overlapping data between adjacent FTP partitions. However, they did not take into account weight partitioning. This is an important factor because it is not possible to store a large amount of weight data on a single resource-constrained edge device. On the basis of Zhao et al. (2018b) and Stahl et al. (2021) considered weight partitioning. In addition, they proposed an integer linear programming (ILP) approach to find the optimal CNN partition thus minimizing the communication requirements.

Unlike the above works, Lane et al. (2016) partitioned and assigned the network to local heterogeneous processors. They proposed two strategies: runtime layer compression (RLC) and deep architecture decomposition (DAD). First, they compressed the DNN with SVD. Then, the compressed model was partitioned and assigned to specific processors for computing.

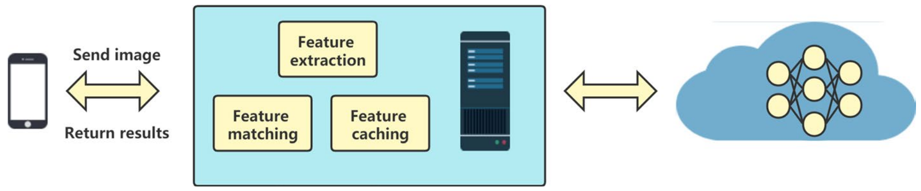


Fig. 13 The illustration of computation results cache

5.3.2 Edge caching

There are two types of redundancy in edge environments (Xu et al. 2020): data redundancy and computation redundancy. Data redundancy is caused by users repeatedly accessing data with high prevalence, while computation redundancy is caused by users requesting the same service. Redundancy can be mitigated by caching data or computation results in edge servers. There exists a large amount of research in edge caching, including cache deployment, cache replacement, etc. Here, we focus on the application of edge caching in edge intelligence. According to the nature of the works, we classify these efforts into three categories: (1) data cache, (2) computation results cache, and (3) research on caching mechanisms.

Data cache A large amount of research in edge caching has been focusing on real-time mobile vision. These works usually utilize DNN to accurately detect the object in the initial frame. Then the tracking algorithm is used to track subsequent frames that are cached locally. Glimpse Chen et al. (2015a) first applied the cache techniques to DNN inference. Glimpse selectively sends some key frames to the server, which marks the objects in the key frames. The local device caches a portion of the subsequent frames. Then the target objects in the subsequent frames are tracked based on the results returned by the server. To enable fast detecting targets to maintain tracking accuracy, (Liu et al. 2019) designed a dynamic regions of interest (ROI) encoding technique and a parallel stream and inference method to reduce transmission delay. In addition, they proposed a moving vector-based target tracking technique.

All the efforts mentioned above offload some tasks to the cloud or edge servers. However, this is accompanied by the transmission delay. Different from them, MARLIN (Apicharttrisorn et al. 2019) does not offload tasks to the cloud or edge. It contains two parts: a target detector and a target tracker. First, MARLIN determines the location of targets based on the initial input frame. Then subsequent frames are cached in the target tracker and the targets are continuously tracked in the subsequent frames. DNN is re-triggered when the targets in the frames change significantly. However, in MARLIN, the target detector and target tracker run sequentially, which is not suitable for scenarios where the video content changes frequently. Different from MARLIN, AdaVP (Liu et al. 2020) runs the target detector and the target tracker in parallel. Among them, the target detector corrects the error of the target tracker. They designed two models as target detectors and dynamically selected a model for tracking based on the environment.

Computation results cache Since there are limited targets in the same environment, the images captured by mobile devices are often very similar. The core idea of computation-results-cache is that image features are stored at the local edge server. Therefore, when the local device requests the DNN service, the server only needs to match the image features and return results (Fig. 13).

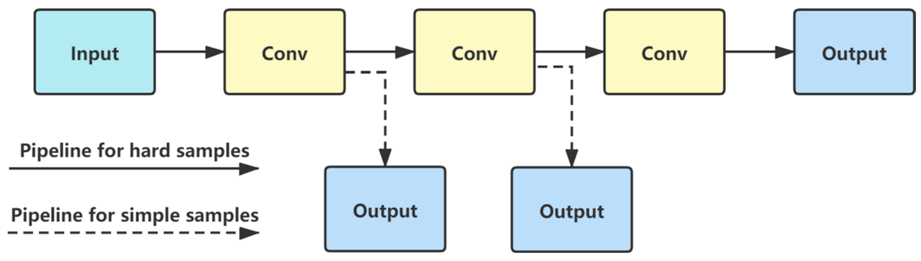


Fig. 14 The illustration of model early exit

Drolia et al. (2017b) cached the computation results at the edge server. As shown in Fig. 13, when a user requests DNN services, the edge device uploads the captured images to the edge server. Then, the server extracts the image features and finds the best match in the cache. For further accelerating inference, Drolia et al. (2017a) suggested utilizing the available resources of the device to reduce the communication latency. Specifically, they used a Markov model to predict the user's next request. The computation results are stored on the device in advance. In addition, they could dynamically adjust the parameters and cache contents based on the results predicted by the Markov model.

Research on caching mechanisms To reuse computation results more efficiently, a portion of the research focused on designing schemes and mechanisms for edge caching. To enable reusing computation results according to the similarity of the input information, Guo et al. (2018b) proposed two schemes: adaptive locally sensitive hashing (A-LSH) and homogenized kNN (H-kNN). The former enables fast and constant lookup, and the latter guarantees highly accurate computation reuse. Xu et al. (2018) proposed a new caching mechanism. They used the input video frames as keys and the reusable regions in feature maps as values. During the CNN runtime, the cached reusable regions are used to replace the CNN computation.

5.3.3 Conditional computation

The main idea of conditional computation is to selectively perform DNN inference for reducing computation. We divide this part of the works into three parts: model early exit, input filtering, and model selection.

Model early exit The key to model-early-exit is determining when the sample exits from the network. The concept of model-early-exit was first introduced in Teerapittayanon et al. (2016). As shown in Fig. 14, they added several branches at specific layers. When the confidence of the inferred result is high enough, the sample is allowed to exit from the network early through the branches. They used the entropy of the classification result to determine whether the sample can exit from the network. When the entropy is less than a pre-defined threshold, it means that the inferred result is trustworthy. At this point, the sample exits from the network. Different from the work of Teerapittayanon et al. (2016), Panda et al. (2017) compared the maximum confidence of the prediction with the threshold. If the confidence is greater than the threshold, the prediction is considered as reliable. Similarly, Laskaridis et al. (2020) also used the comparison of confidence and the threshold for determining if the sample should exit from the network. The authors combined model-early-exit

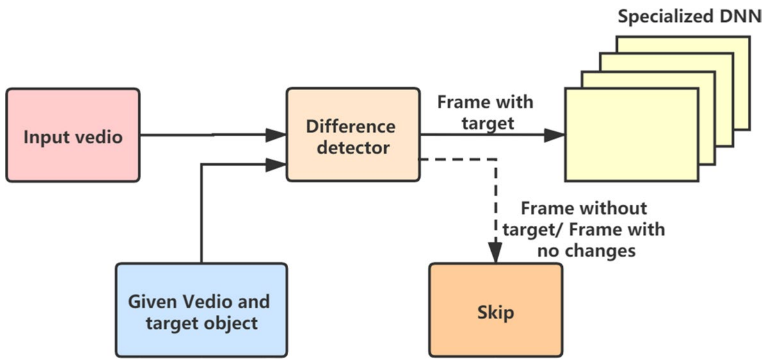

Specialized DNN
Input vedio
Difference detector
Frame with target
Frame without target/ Frame with no changes
Given Vedio and target object
Skip

Fig. 15 The workflow of Noscope

with model partition. They set up an analyzer and a dynamic scheduler to find the optimal partition points and exit points. First, the analyzer analyzes the metrics of the model. Then the dynamic scheduler selects an optimal result based on the current situation. Finally, the results that prioritize satisfying the most important optimization objective are found according to the importance of the objectives.

The thresholds in the works mentioned above are user-defined, while (Lo et al. 2017) defined thresholds using a statistical method. They added an Authentic Operation (AO) unit to the network. The AO unit sets different thresholds for different DNN output classes. These thresholds determine whether the samples need to be transferred to the server for further inference. Moreover, they set up corresponding strategies according to the stability of the channel. When the channel is unstable, the input samples are traversed across the entire network at the edge for maintaining accuracy. Otherwise, more workload is transferred to the main network at the cloud.

Laskaridis et al. (2020), Lo et al. (2017), Panda et al. (2017) and Teerapittayanon et al. (2016) determined whether the current sample needs to be further inferred by confidence and threshold. Unlike the above works, Bolukbasi et al. (2017) introduced a decision function for determining when the current sample exits from the network. Moreover, they introduced model selection to make trade-offs in terms of inference time and accuracy.

Although DNN model with multiple exit points can dynamically select the best exit point for each sample, each exit point incurs additional computation complexity. Lee et al. (2020) investigated how the number of exit points affects model performance. They found that when using models with a single exit point, higher accuracy can be obtained with fewer calculations.

Input filtering Similar to the model-early-exit, input filtering speeds up inference by selectively processing data. According to the difference of these works, we divide the efforts into two categories: (1) Filtering of video data; (2) Filtering in edge offloading.

(1) Filtering of video data As shown in Fig.15, Noscope (Kang et al. 2017a) contains a specialized DNN and a difference detector. The specialized DNN abandons the generalization capability and performs inference only in the specified task. The difference detector checks the differences between frames. Frames with little or no change are skipped to speed up inference. Hsieh et al. (2018) proposed a two-stage filtering system, Focus. In the input stage, Focus uses a lightweight CNN to classify the objects.

Then, similar objects are clustered and the top-k results are used to index the cluster centers. In the query phase, Focus uses a complicated CNN for further inference and return frames containing the query target. Zhang et al. (2018a) and Zhang et al. (2020) proposed a three-stage filtering system, FFS-VA. It contains three models: a stream-specialized difference detector (SDD), a stream-specialized network model (SNM) and a Tiny-YOLO-Voc model. Among them, the SDD is used to remove frames that only contain background. The SNM is used to identify frames that contain target objects. The Tiny-YOLO-Voc is used to filter frames where the confidence of target objects is less than a pre-defined threshold. The above methods only filtered the input of a single video. Unlike the above methods, Jain et al. (2020) and Jain et al. (2018) built a cross-camera spatio-temporal correlation model from unlabeled video data. In the inference phase, they used this model to filter frames that have no spatio-temporal correlation with the current location of the query entity, thus reducing inference.

(2) Filtering in edge offloading Besides the above efforts, some works applied input filtering in edge environments to determine which data needs to be offloaded. Wang et al. (2018a) used MobileNet and SVM to filter irrelevant frames on Unmanned Aerial Vehicles, thus saving bandwidth when offloading. However, they did not optimize for the multi-tenant problem. In a further step, FilterForward (Canel et al. 2019) set up micro-classifiers (MCs) at the edge to filter irrelevant frames. Moreover, FilterForward can run tens of micro-classifiers simultaneously, which compensates for the shortcoming of Wang et al. (2018a). The works mentioned above performs filtering at the edge. Pakha et al. (2018) allowed the server to decide which frames to be filtered. In this work, the client sends images with different resolution to the server, which processes the images and returns the results to the client. Specifically, the client sends a low-resolution frame to the server. If the target object is not detected in the low-resolution frame, the frame is filtered. On the contrary, if the target object is detected with a certain degree of confidence, the regions of interest are cropped and sent to the server at a high resolution.

Model selection The main idea of model selection is to adaptively select the most suitable DNN for inference tasks according to factors such as inference difficulty, accuracy requirements and inference time. According to the nature of these efforts, we divide the efforts in this area into two categories: (1) Selection of independent model; (2) Selection of partial model.

(1) Selection of independent model Park et al. (2015) is the earliest work on model selection. For a given sample, they first performed inference with a light-weight DNN. When the confidence is lower than a pre-defined threshold, a complex DNN is used for inference. Stamoulis et al. (2019) treated model selection as a hyperparameter optimization problem under hardware constraints and solved it with Bayesian optimization. Marco et al. (2019) and Taylor et al. (2018) trained a predictor with machine learning. For given input samples, it is able to dynamically select the optimal DNN according to the desired accuracy and inference time. MobiSR Lee et al. (2019) generates a lighter super-resolution (SR) model based on the original SR model with model compression. Then, an appropriate model is selected according to the difficulty of upgrading the low-resolution input image and the load of the computing engine. Yang et al. (2020b) proposed RANet, an adaptive network, which contains subnets with different input resolution. The samples are identified starting from the smallest subnet and can be

exited early if the confidence of the sample is greater than the threshold. Otherwise, the next-level subnet fuse the features of the current subnet and continue to identify samples.

(2) Selection of partial model All of the works mentioned above select independent model for inference. Although this approach can speed up inference, the total size of the models and the training time also increase significantly. To alleviate this problem, Tann et al. (2016) used incremental learning to train the network and dynamically adjusted the channels in the network according to multiple factors such as response time, power, and accuracy targets. They devised a method to determine the appropriate network width and dynamically adjust the network when inference errors occur. Unlike the above works, only the weights of a complete network need to be stored in this work. Similarly, Panda et al. (2016) constructed a tree of classifiers, where the shallow classification network classifies the common features and selectively activates subsequent networks based on the features of the samples. Wang et al. (2018b) proposed SkipNet. They added gating modules to the network, which maps the activation of the previous layer as a binary to decide whether to skip the next layer.

5.3.4 Open-source frameworks for inference accleration

(1) *TensorRT* TensorRT[5] is a high-performance inference framework proposed by NVIDIA that enables well-trained DL models to be deployed on NVIDIA GPUs with low latency and high throughput. TensorRT is a C++ library that provides C++ API and Python API. In addition, it supports mainstream DL frameworks such as Caffe, Tensorflow, Pytorch, MxNet, etc. It is necessary to obtain a runtime engine with TensorRT in order to deploy a trained model. It is worth noting that for network models built with Caffe and Tensorflow, TensorRT maps the layers in them and optimizes them for NVIDIA's GPUs. However, models trained with other frameworks have to be first converted to a generic ONNX model and then parsed by TensorRT. ONNX is a file format designed for machine learning to store trained models. It enables different DL frameworks to store models in the same format.

(2) *Tensorflow Lite* Tensorflow Lite[6] is a lightweight solution for embedded and mobile devices proposed by Google. With Tensorflow Lite, developers can run Tensorflow models on embedded, mobile and IoT devices. Tensorflow Lite is composed of two main components: (1) Tensorflow Lite interpreter: it allows optimized models to be run on different types of hardware such as cell phones and embedded devices. (2) Tensorflow Lite Converter: it reduces binary file size and improves performance, thus optimizing the model for the interpreter. In addition to model-specific optimizations, Tensorflow Lite supports the Andriod neural network API to take full advantage of available hardware accelerators of mobile devices. When the hardware accelerator is unavailable, Tensorflow Lite uses the CPU for computation, ensuring that the model can still be run on a large number of devices.

(3) *Paddle Lite* Paddle Lite[7] is a high-performance, lightweight, flexible and easy-to-extend DL framework that supports multiple hardware platforms including mobile, embedded and server-side. It is proposed by Baidu and provides three APIs including C++, Java,

5 https://github.com/NVIDIA/TensorRT.

6 https://github.com/tensorflow/tflite-micro.

7 https://github.com/PaddlePaddle/Paddle-Lite.

and Python. Paddle Lite optimizes machine learning on mobile devices, compressing model and binary file sizes and accelerating inference. It is worth noting that Paddle Lite directly supports the model file format saved by PaddlePaddle. For models built with other DL frameworks (e.g. Tensorflow, Pytorch), X2Paddle[8] needs to be used to convert the model files to PaddlePaddle format. Paddle Lite supports multiple types of computational graph optimization including operator fusion, computation pruning, memory optimization, and quantization computation. In addition, it also supports hybrid scheduling between different hardware, which can make full use of various hardware resources.

6 AI appllications on the edge

In the above sections, we have introduced the techniques, architectures and open-source frameworks of edge training and edge inference. In this section, we introduce the efforts of edge training and edge inference from an application perspective. We summarize these efforts in Table 6.

6.1 Training models on the edge

6.1.1 FL for healthcare

The development of DL has brought revolutionary changes in disease diagnosis. However, a large amount of data is necessary for getting a well-trained DL model. However, it is difficult to share data among various medical institutions due to privacy issues. FL can get a well-trained model while protecting patients' privacy. Thus, it can facilitate the development of AI in the medical field. For example, with the help of FL, Sheller et al. (2018) trained a brain tumor segmentation model with data from multiple medical institutions. The model performs similarly to a model trained with centralized data. Further, to avoid the privacy threat posed by PS. Roy et al. (2019) proposed BrainTorrent, a peer-to-peer FL framework for medical collaboration. In addition, they found that the server-less BrainTorrent approach outperforms the traditional server-based approach.

6.1.2 FL for recommendation system

Traditional recommendation system models are trained with centrally stored user behavior data. However, centralized data storage may lead to privacy issues. FL can provide users with personalized recommendations while protecting their privacy. For example, by using FL, Qi et al. (2020) trained a news recommendation model while keeping private data on the user's device. In addition, they further protected user privacy by adding noise to the gradients. Extensive experiments on real-world datasets showed that the method can achieve a performance comparable to that of the state-of-the-art news recommendation methods at that time. Similarly, with the help of FL, Hartmann et al. (2019) improved the suggestion ranking of Firefox URL without collecting users' data, which allows the users to type about half a character less before selecting an item.

[8] https://github.com/PaddlePaddle/X2Paddle.

Table 6 The main efforts of AI applications on the edge

Related work	Type	Architecture	Application domain	Technology used
Sheller et al. (2018)	Edge training	PS architecture	Healthcare	Federated learning
Roy et al. (2019)	Edge training	Gossip architecture	Healthcare	Federated learning
Qi et al. (2020)	Edge training	PS architecture	Recommendation system	Federated learning
Hartmann et al. (2019)	Edge training	PS architecture	Recommendation system	Federated learning
Zhang et al. (2021a)	Edge training	PS architecture	Industrial internet of things	Federated learning
Liu et al. (2021)	Edge training	PS architecture	Industrial internet of things	Federated learning
Savazzi et al. (2020)	Edge training	Gossip architecture	Industrial internet of things	Federated learning
Samarakoon et al. (2020)	Edge training	PS architecture	Vehicular network	Federated learning
Chen et al. (2015a)	Edge inference	Device-cloud co-inference	Object tracking	Model partition
Liu et al. (2019)	Edge inference	Device-edge co-inference	Object tracking	Model partition
Du et al. (2020b)	Edge inference	Device-cloud co-inference	Object tracking	Model partition
Wang et al. (2021)	Edge inference	Device-edge co-inference	Object tracking	Model partition
Li et al. (2018c)	Edge inference	Device-edge-cloud co-inference	Industrial internet of things	Model partition & model-early-exit
Zeng et al. (2019)	Edge inference	Device-edge co-inference	Industrial internet of things	Model partition & model-early-exit
Zhou et al. (2019a)	Edge inference	Co-inference between devices	Industrial internet of things	Model partition
Stahl et al. (2021)	Edge inference	Co-inference between devices	Industrial internet of things	Model partition
Wang et al. (2018a)	Edge inference	Device-edge co-inference	Continuous mobile vision	Model partition and input filtering
Canel et al. (2019)	Edge inference	Device-edge co-inference	Continuous mobile vision	Model partition and input filtering
Yi et al. (2020)	Edge inference	Device-cloud co-inference	Continuous mobile vision	Model partition
Chen et al. (2019a)	Edge inference	Device-only	Smart driving	Lightweight model design
Dwisnanto Putro et al. (2020)	Edge inference	Device-only	Smart driving	Lightweight model design
Drolia et al. (2017b)	Edge inference	Edge-only	Object recognition	Edge caching
Zeng et al. (2017)	Edge inference	Device-only	Object recognition	Knowledge distillation
Ling et al. (2020)	Edge inference	Device-only	Object recognition	Lightweight model design
Bhattacharya and Lane (2016)	Edge inference	Device-only	Human activity recognition	–
Almaslukh et al. (2018)	Edge inference	Device-only	Human activity recognition	Lightweight model design

6.1.3 FL for industrial internet of things

Training AI models usually requires large amounts of high-quality labeled data. However, it is often difficult and expensive to collect data in industrial scenarios. Obviously, collecting data from different clients to train the model is a good solution. However, data sharing is not widely used in practical industrial scenarios due to potential conflicts of interest. FL can get a well-trained model without leaking clients' data. Therefore, some existing works used FL to solve the data island problem that exists in industrial scenarios. For example, Zhang et al. (2021a) proposed a FL-based approach for mechanical fault diagnosis. Specifically, clients train local models, and the server aggregates local models to obtain a global model. Gradients rather than data are exchanged between clients and the server. In addition, they proposed a self-supervised learning strategy to improve the performance of the model.

Edge devices failing can have a serious impact on industrial production. Moreover, edge devices often contain private information about users. In order to detect anomalous devices while protecting user privacy, Liu et al. (2021) proposed an on-device FL-based anomaly detection framework for sensing time-series data in IoT environments. In addition, they proposed a gradient compression strategy based on top-k selection for improving communication efficiency. The framework detects anomalies accurately and timely while reducing communication overhead by 50% compared to FL without a gradient compression policy.

The two efforts mentioned above utilizes the PS architecture-based FL. Different from them, (Savazzi et al. 2020) revisited the FedAvg algorithm in the context of industrial IoT and proposed several FedAvg algorithms based on the Gossip protocol, which provide reliable robustness and scalability.

6.1.4 FL for vehicular networks

The mission of vehicular network requires a huge amount of road test data, which is transmitted to the cloud for processing. However, the large amount of data transmission can cause communication delays. In addition, there are a long business process and complex environments in vehicle networks. Traffic conditions vary greatly from location to location. Therefore, the development of secure transportation is limited due to the data island problem. Samarakoon et al. (2020) applied FL to solve joint power and resource allocation problems in vehicular networks. First, vehicles train local models and transmit them to the roadside units. The roadside units aggregate the local models transmitted by each vehicle to obtain a global model and then return it to the vehicles. With FL, vehicles can predict the queue tail distribution locally without sharing actual queue length samples, which reduces unnecessary overheads. Based on the queue information, resource allocation can be optimized better. The method's accuracy is comparable to that of a centralized solution. Moreover, the amount of data exchanged can be reduced by 79%.

6.2 Model inference on the edge

6.2.1 Collaborative inference

(1) *Co-inference for object tracking* The development of DL has greatly enhanced the accuracy of object tracking. However, due to the computationally intensive nature

of DL, it is difficult to perform target tracking tasks on resource-constrained edge devices. Although the cloud or edge can provide arithmetic support to the devices, cloud-based or edge-based inference suffers from unbearable communication latency. Chen et al. (2015a) utilized cloud-device co-inference for target tracking. Specifically, they performed target tracking algorithms at the device and maintained accuracy with a cloud-based object detection model. Further, Liu et al. (2019) designed a system and proposed dynamic ROI encoding techniques and a parallel stream inference method to further reduce latency. The system can improve detection accuracy by 20.2–34.8%, while only 2.24 ms latency is required for target tracking on AR devices. The two studies mentioned above do not optimize small object detection. Du et al. (2020b) presented DNN-Driven Streaming (DDS). Specifically, DDS allows the server to determine the areas of the frame that need to be re-encoded. Then, the camera recodes the areas containing small objects and offloads them to the server for inference. Compared to the solutions available at the time, DDS maintains higher accuracy and reduces bandwidth usage by 59%. Although DDS improves the small object detection accuracy, it increases the latency of object detection. To alleviate this problem, Wang et al. (2021) presented EdgeDuet. Specifically, EdgeDuet performs large object detection on low-resolution video frames with a lightweight model, while offloading high quality video frames for small object detection. EdgeDuet offers a 233% improvement in accuracy over local object detection on small object detection tasks. In addition, it improves total latency by 44.7% and end-to-end latency by 34.2% over the state-of-the-art offloading strategy.

(2) *Co-inference for industrial internet of things* With the rapid development of IOT and network infrastructures, there are a large number of sensors in industrial scenarios, resulting in a surge in data volume. If all these data are offloaded to the cloud for processing, it would put a huge burden on network bandwidth and servers. Therefore, it is necessary to offload some data to the edge for processing. Some existing works utilize co-inference between device-edge-cloud to ease the pressure on data centers. Li et al. (2018c) designed DeepIns, a manufacturing inspection system. To enable real-time data processing, they used edge servers and cloud servers as exit points, speeding up response time and reducing network traffic. Zeng et al. (2019) combined model partition and model-early-exit and proposed Boomerang. They found optimal partition points and exit points with deep reinforcement learning, achieving maximum accuracy while meeting latency. The efforts mentioned above utilize cloud-edge-device co-inference to accelerate inference. Unlike them, Zhou et al. (2019a) proposed AAIOT, which can allocate inference tasks to each device in multi-layer IoT systems, accelerating inference through collaboration between devices. Similarly, Zhao et al. (2018b) proposed DeepThings, a framework that can adaptively allocate inference tasks on resource-constrained IoT devices. DeepThings enables 1.7×–2.2× inference acceleration on 2–6 edge devices with less than 23 MB memory.

(3) *Co-inference for continuous mobile vision* There are a huge number of cameras in modern cities, which are generating a huge amount of video data. DL-based video analysis can significantly improve the efficiency of video processing. However, uploading such a huge amount of data to the cloud would put a tremendous strain on the network infrastructure. Obviously, how to perform video analysis efficiently and quickly has become an important issue. Wang et al. (2018a) utilized MobileNet and SVM to filter irrelevant video frames on drones, thus reducing network bandwidth consumption and accelerating video analysis. As a further step, Canel et al. (2019) proposed FilterForward. Specifically, they utilized small neural networks rather than SVMs to determine whether frames should be filtered, which improves the accuracy of filter-

ing. In addition, they optimized for the multi-tenancy problem. FilterForward reduces bandwidth usage by an order of magnitude while maintaining accuracy. In order to be able to find missing people in large, crowded cities, Yi et al. (2020) designed EagleEye, which contains several complex DNNs including face detection network, resolution enhancement network, and face recognition network. To provide low latency services, they developed Content-Adaptive Parallel Execution, using devices and cloud to optimize multi-DNN model execution pipeline. Compared to naïve execution, EagleEye reduces latency by 9.07x while offloading only 108k of data.

6.2.2 Solo inference

(1) *Solo-inference for smart driving* Due to transmission latency, cloud-based or edge-based inference cannot provide real-time detection for smart driving. Therefore, it is necessary for devices to participate in inference. There are many efforts that applied the works mentioned in the above sections for smart driving. For example, to assist people with driving, Chen et al. (2019a) presented a shallow model called concatenated feature pyramid network for real-time embedded traffic flow prediction. They proposed concatenated blocks to reduce the number of convolutional layers and connections. Dwisnanto Putro et al. (2020) proposed an eyes detector that can be run in real time on the CPU for detecting driver's fatigue states. It consists of two modules, a backbone network for feature extraction and a detection module for predicting eye regions. The eyes detector can reach a detection speed of 467 frames per second on the Intel-I5-6600.

(2) *Solo-inference for object recognition* Since there are limited objects in the edge environments, duplication of recognition tasks in the same scene may occur. Drolia et al. (2017b) cached the computation results in the edge server. When the client sends images to the edge server, the edge server performs feature extraction and feature matching on the sent images. For objects that are already cached in the edge server, recognition results are returned directly. To be able to accurately identify prescription pills on mobile devices, Zeng et al. (2017) developed a pills identification system named MobileDeepPill. They utilized a multi-CNNs model to collectively capture the shape, color, and imprints features of the pills. In addition, for speeding up the inference, they utilized a knowledge distillation-based model compression technique to reduce the number of parameters and FLOPs of the multi-CNNs model. However, MobileDeepPill simply extracts different features without considering the complementary relationship between different features. Ling et al. (2020) proposed two training strategies, Batch ALL and Batch Hard, to allow different features to compensate each other for better recognizing hard samples.

(3) *Solo-inference for human activity recognition* With the popularity of smart devices, there has been a tremendous growth in the data collected by smart devices. However, these data contain the personal information of the user. To protect user privacy, it is necessary to process the data from smart devices locally. Almaslukh et al. (2018) designed an efficient and lightweight model for smartphone-based human activity recognition. They used time-domain statistical features to extract more distinguishable features. In addition, they explored a new data augmentation method, which can improve the accuracy of smartphone-based human activity recognition. Sensors collect a large amount of data in the edge environment. However, these data are difficult to interpret and label. For providing human activity recognition labels to sensor data, Radu and

Henne (2019) proposed a framework, Vision2Sensor, which synchronizes the visual images captured by the camera with the sensor data to generate labels. The system can be run on embedded devices in real time, thus protecting user privacy.

7 Open challenges and future directions

From the discussion above, we can see that the benefits of edge intelligence are obvious. It relieves the pressure on data centers and brings intelligent services down to the edge. However, there are still many challenges on the way to achieve edge intelligence. In this section, we present open challenges, possible solutions and future directions for edge intelligence.

7.1 The specialties of data in edge environments

Unlike the data in data centers, data in edge environments often have the following characteristics: (1) Persistence of data; (2) Unlabeled data; (3) Non-IID data; (4) Data with noise. We provide possible solutions for the specialties of data in edge environments.

(1) Persistence of data: in the edge environment, data is continuously generated. For supervised learning, the models are trained with pre-collected static datasets and cannot learn new knowledge when the training is over. Clearly, models cannot be retrained on resource-constrained edge devices. In addition, it is not possible to get all the training data at once in edge environments. For this situation, we believe that incremental learning may be a solution. Actually, there are already some efforts to explore online incremental learning on edge devices, which requires less computational resources than training models from scratch. In addition, models trained with incremental learning are able to perform equally well on new tasks and old ones. For example, Li et al. (2019a) proposed the RILOD system, which progressively trains an existing object detection model so that it can detect new targets without compromising its original capabilities. In addition, they designed a real-time dataset construction pipeline for collecting training images and automatically labeling images. Experiments have shown that RILOD can learn to detect a new object in a few minutes. With incremental learning, the problem of persistence of edge data can be solved to a great extent. In addition, the model can be extended without hurting its original capabilities. However, there are still some challenges with incremental learning on edge devices, such as hardware. Traditional hardware platforms such as CPUs and GPUs are not able to handle DNNs on edge devices efficiently. CPU cannot be used for parallel computing and therefore is computationally inefficient. GPU can provide high throughput but its low energy efficiency prevents it from being deployed on edge devices. In fact, there has been some efforts to design efficient hardware architectures. For example, (Luo and Yu 2021) proposed a compute-in-memory-based accelerator, AILC, for on-chip incremental learning using spin-transfer-torque magnetic random access memory (STT-MRAM) technology.

(2) Unlabeled data: for supervised learning, a complete labeled dataset is required for training. However, data in edge environments is often unlabeled. Active learning may be a solution to alleviate this problem. It allows learning algorithms to proactively make annotation requests and hand over the filtered data to human experts for further annotation. Although it can reduce the costs of data annotation, human annotations are still needed. In fact, several works have attempted to utilize active learning to solve

the problem of unlabeled data in edge environments. For example, Diethe et al. (2016) proposed hierarchical Bayesian active learning methods to label data in smart home. Shahmohammadi et al. (2017) used active learning to label the sensor data collected by the smartwatch for human activity recognition. In addition, unsupervised learning is also an optional solution. Unlike supervised learning, it is able to learn from unlabeled data. Janjua et al. (2019) proposed IRESE, a rare-event detection system that processes input data utilizing unsupervised machine learning. It achieves over 90% accuracy in detecting audio rare events such as gunshot, scream, and siren.

(3) Non-IID data: models trained with IID data can learn the data distribution characteristics well. However, the data generated by edge devices is usually Non-IID, which means the data distribution of each edge device cannot represent the total data distribution. A portion of existing research has been focusing on how to process Non-IID data in FL. For example, Zhao et al. (2018a) suggested creating a subset of data that is shared globally to improve the training on Non-IID data. Yoshida et al. (2020) assumed that a few clients allow their data to be uploaded to the server in order to form an approximate IID dataset. The collected IID dataset was utilized to train a model. Then the model is aggregated with other local models trained with Non-IID data.

(4) Data with noise: for IoT devices in different edge environments, the noise in environments may affect the data and consequently the model performance. We believe that input filtering may be a good solution. For example, Li et al. (2020) filtered invalid data in IoT environments, thus avoiding its impact on sentiment detection. However, when there is too much noise in the edge environment, a small number of high-quality labeled samples may not be enough to train a model. In this situation, we believe that the following technologies may be potential solutions. First, data augmentation, which can enhance the generalization ability of the network by increasing the number of samples. For example, Mathur et al. (2018) used data augmentation to address noise pollution caused by sensor heterogeneity. Second, few-shot learning, which can improve the accuracy when the number of samples in a category is small. For example, Lungu et al. (2020) proposed an improved Siamese Networks for few-shot learning. In a 5-way, 1-shot classification task, this network improves accuracy by up to 22% on average across four datasets.

7.2 Privacy and security issues

For cloud-based DL services, data needs to be uploaded to the cloud for inference, which can be a threat to user privacy. Moreover, for organizations like medical centers, they need to protect the privacy of patients. Therefore, it is not possible to exchange data with other organizations to train models. FL can be a good solution to this problem. For example, Sheller et al. (2018) used data from multiple institutions to train a brain tumor segmentation model without sharing data. However, one of the disadvantages of FL with PS architecture is the dependence on PS. For application scenarios like smart home, it is not difficult to find a trusted third party as PS. However, for collaboration between medical centers, it is difficult to find a trusted third party (Roy et al. 2019). Decentralized FL can be a good solution to this problem, which is mainly achieved through blockchain (Weng et al. 2021; Kim et al. 2019) or other communication methods (Lu et al. 2020; Hu et al. 2019). For example, Roy et al. (2019) proposed a peer-to-peer FL framework for medical collaboration, which does not require a trusted third party to aggregate local models.

7.3 Scheduling and trade-offs of DNN services

There are often multiple DNN service requests in edge enviroments. For example, requests from vehicles in autonomous driving for services such as object detection and path planning. In this case, multiple DNN programs compete for limited resources. Therefore, service providers need to optimize the scheduling of DNN services. There have been works exploring the DNN scheduling problem in edge environments. For example, Huang et al. (2019) designed an efficient heuristic online algorithm for multitask scheduling.

In addition, various factors such as model accuracy, inference time need to be traded off in order to reduce resource consumption. We believe that model selection and model-early-exit are optional solutions. The former selects the most suitable model by measuring various factors such as accuracy and inference time (Taylor et al. 2018). This method is suitable for devices with large storage such as smartphones since the weights of multiple models need to be stored. Similarly, the latter determines which layer to exit for samples by measuring the difficulty of inference (Teerapittayanon et al. 2016). The difference is that for the latter, only the weights of one model need to be stored, which makes it suitable for devices with small storage. In fact, quite a few efforts have been made to use both techniques in edge environments to achieve the accuracy-inference time trade-off. Bolukbasi et al. (2017) utilized model selection to make a trade-off in terms of inference time and accuracy. Zeng et al. (2019) achieved maximum accuracy while meeting the inference time by utilizing model partition and model-early-exit.

7.4 Incentives and penalties

Many existing works on FL suffer from several problems: (1) device resources are inevitably consumed when users participate in collaborative training, which may lower the willingness of users to engage in training; (2) users, as providers of the data, may refuse to participate in training due to privacy concerns; (3) many of existing works on FL considered only privacy threats from PS (Bonawitz et al. 2017; Geyer et al. 2017; Truex et al. 2019; Wei et al. 2020). However, there are likely to be malicious nodes during the training process, which can seriously disrupt the training. Moreover, in FL, local nodes have absolute control over their own data. Therefore, anomaly detection cannot be used for checking malicious nodes. Blockchain can be a promising solution for the above problems. For example, in FLChain (Bao et al. 2019), honest users can earn more partition profits with a well-trained model, and malicious users are promptly detected and severely penalized.

7.5 The development of training models dynamically

In order to deploy the model to edge devices, two training methods can be used: dynamic training and static training. Among them, static training means that once training is complete, the model cannot be fine-tuned. It is dependent on high performance computing nodes. Dynamic training refers to training with computation resources of edge nodes. Due to the specificity of the data generated at the edge, models obtained by static training may not be available for edge environments all the time. On the contrary, the model obtained by dynamic training can be applied to the edge environments well. Moreover, when the data in the environment changes, the model can be adjusted by fine-tuning. It is obvious that the development of dynamic training plays a crucial role in the implementation of edge intelligence.

7.6 The development of cloud-edge-device collaborative inference

A part of the existing works (Mao et al. 2017a, b; Zhou et al. 2019a) performed inference completely on devices for avoiding transmission latency. We believe that edge computing, as an extension of cloud computing, can provide low latency to the cloud. Moreover, the cloud can provide computing support to the edge. The collaboration between cloud, edge and devices can achieve a great trade-off between latency, inference accuracy and energy consumption. Especially in the future 6G era, powerful network bandwidth and communication capability can provide strong support for the collaboration between cloud, edge and devices.

7.7 The extension of edge intelligence

From the discussion in the above sections, we can see that edge intelligence can provide better services in several aspects such as user privacy, communication bandwidth, and transmission latency. However, intelligent services can be deployed on more fine-grained devices. Tiny machine learning (TinyML) (Sanchez-Iborra and Skarmeta 2020; Banbury et al. 2020), as an extension of edge intelligence, can be deployed on micro-controllers. This intelligence paradigm can provide more reliable intelligent services. Although research on TinyML is still in its initial stage, we believe that only through the collaboration between cloud-based intelligence, edge intelligence and TinyML can we effectively build a complete intelligence eco-system.

8 Conclusion

Edge computing, as the extension of cloud computing, is promising to bring compute-intensive DL services down to the edge. The combination of AI and edge computing has produced a new paradigm, edge intelligence, which is gradually attracting the attention of researchers in academia and industry. In this paper, we provided a detailed introduction of techniques, architectures, frameworks and implementations for edge training and edge inference, and compared their advantages and disadvantages from multiple perspectives. In addition, we discussed the challenges, possible solutions and future directions of edge intelligence. We believe that in the near future, with the rapid development of AI and edge computing, more and more researchers will be dedicated to implementing edge intelligence.

Acknowledgements The authors are very appreciative to the reviewers for their precious comments which enormously ameliorated the quality of this paper. This work was supported in part by National Key R&D Program of China (2018YFB1701802); National Natural Science Foundation of China 61802280,61806143, 61772365, 41772123; Tianjin Technology Innovation Guide Special 21YDTPJC00130.

References

AbdulRahman S, Tout H, Mourad A et al (2020) Fedmccs: multicriteria client selection model for optimal iot federated learning. IEEE Internet Things J 8(6):4723–4735. https://doi.org/10.1109/JIOT.2020.3028742

Aji AF, Heafield K (2017) Sparse communication for distributed gradient descent. ArXiv preprint arXiv:1704.05021

Almaslukh B, Al-Muhtadi J, Artoli AM (2018) A robust convolutional neural network for online smartphone-based human activity recognition. J Intell Fuzzy Syst 35(2):1609–1620. https://doi.org/10.3233/JIFS-169699

Anwar S, Sung W (2016) Compact deep convolutional neural networks with coarse pruning. ArXiv preprint arXiv:1610.09639

Anwar S, Hwang K, Sung W (2017) Structured pruning of deep convolutional neural networks. ACM J Emerging Technol Comput Syst (JETC) 13(3):1–18. https://doi.org/10.1145/3005348

Aono Y, Hayashi T, Wang L et al (2017) Privacy-preserving deep learning via additively homomorphic encryption. IEEE Trans Inf Forensics Secur 13(5):1333–1345. https://doi.org/10.1109/TIFS.2017.2787987

Apicharttrisorn K, Ran X, Chen J, et al (2019) Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In: Proceedings of the 17th conference on embedded networked sensor systems, pp 96–109. https://doi.org/10.1145/3356250.3360044

Astrid M, Lee SI (2017) Cp-decomposition with tensor power method for convolutional neural networks compression. In: 2017 IEEE international conference on big data and smart computing (BigComp). IEEE, pp 115–118. https://doi.org/10.1109/BIGCOMP.2017.7881725

Ba LJ, Caruana R (2013) Do deep nets really need to be deep? ArXiv preprint arXiv:1312.6184

Bagdasaryan E, Veit A, Hua Y, et al (2020) How to backdoor federated learning. In: International conference on artificial intelligence and statistics. PMLR, pp 2938–2948. http://proceedings.mlr.press/v108/bagdasaryan20a.html

Banbury CR, Reddi VJ, Lam M, et al (2020) Benchmarking tinyml systems: challenges and direction. ArXiv preprint arXiv:2003.04821

Bao X, Su C, Xiong Y, et al (2019) Flchain: a blockchain for auditable federated learning with trust and incentive. In: 2019 5th international conference on big data computing and communications (BIGCOM). IEEE, pp 151–159. https://doi.org/10.1109/BIGCOM.2019.00030

Bellman R (1953) An introduction to the theory of dynamic programming. Rand Corporation, Santa Monica. https://apps.dtic.mil/sti/pdfs/AD0074903.pdf

Bengio Y, Ducharme R, Vincent P, et al (2003) A neural probabilistic language model. J Mach Learn Res 3:1137–1155. http://jmlr.org/papers/v3/bengio03a.html

Bhattacharya S, Lane ND (2016) From smart to deep: robust activity recognition on smartwatches using deep learning. In: 2016 IEEE international conference on pervasive computing and communication workshops (PerCom Workshops), pp 1–6. https://doi.org/10.1109/PERCOMW.2016.7457169

Blot M, Picard D, Cord M, et al (2016) Gossip training for deep learning. ArXiv preprint arXiv:1611.09726

Bolukbasi T, Wang J, Dekel O, et al (2017) Adaptive neural networks for efficient inference. In: Precup D, Teh YW (eds) Proceedings of the 34th international conference on machine learning, proceedings of machine learning research, vol 70. PMLR, pp 527–536. https://proceedings.mlr.press/v70/bolukbasi17a.html

Bonawitz K, Ivanov V, Kreuter B, et al (2017) Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp 1175–1191. https://doi.org/10.1145/3133956.3133982

Bonomi F, Milito RA, Zhu J, et al (2012) Fog computing and its role in the internet of things. In: Gerla M, Huang D (eds) Proceedings of the first edition of the MCC workshop on mobile cloud computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012. ACM, pp 13–16. https://doi.org/10.1145/2342509.2342513

Buciluă C, Caruana R, Niculescu-Mizil A (2006) Model compression. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, pp 535–541. https://doi.org/10.1145/1150402.1150464

Bulat A, Tzimiropoulos G (2019) Xnor-net++: improved binary neural networks. ArXiv preprint arXiv:1909.13863

Caldas S, Konečny J, McMahan HB, et al (2018) Expanding the reach of federated learning by reducing client resource requirements. ArXiv preprint arXiv:1812.07210

Canel C, Kim T, Zhou G, et al (2019) Scaling video analytics on constrained edge nodes. ArXiv preprint arXiv:1905.13536. https://proceedings.mlsys.org/book/273.pdf

Chen PY, Hsieh JW, Gochoo M, et al (2019a) Smaller object detection for real-time embedded traffic flow estimation using fish-eye cameras. In: 2019 IEEE international conference on image processing (ICIP), pp 2956–2960. https://doi.org/10.1109/ICIP.2019.8803719

Chen TYH, Ravindranath L, Deng S, et al (2015a) Glimpse: Continuous, real-time object recognition on mobile devices. In: Proceedings of the 13th ACM conference on embedded networked sensor systems, pp 155–168. https://doi.org/10.1145/2972413.2972423

Chen W, Wilson J, Tyree S, et al (2015b) Compressing neural networks with the hashing trick. In: International conference on machine learning. PMLR, pp 2285–2294. http://arxiv.org/abs/1504.04788

Chen W, Wilson J, Tyree S, et al (2016) Compressing convolutional neural networks in the frequency domain. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1475–1484. https://doi.org/10.1145/2939672.2939839

Chen Y, Luo T, Liu S, et al (2014) Dadiannao: a machine-learning supercomputer. In: 2014 47th annual IEEE/ACM International symposium on microarchitecture. IEEE, pp 609–622. https://doi.org/10.1109/MICRO.2014.58

Chen Y, Sun X, Jin Y (2019) Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. IEEE Trans Neural Netw Learn Syst 31(10):4229–4238. https://doi.org/10.1109/TNNLS.2019.2953131

Chollet F (2017) Xception: deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1251–1258. https://doi.org/10.1109/CVPR.2017.195

Courbariaux M, Bengio Y, David JP (2015) Binaryconnect: training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems, pp 3123–3131. http://arxiv.org/abs/1511.00363

Courbariaux M, Hubara I, Soudry D, et al (2016) Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or −1. ArXiv preprint arXiv:1602.02830

Deng S, Zhao H, Fang W et al (2020) Edge intelligence: the confluence of edge computing and artificial intelligence. IEEE Internet Things J 7(8):7457–7469. https://doi.org/10.1109/JIOT.2020.2984887

Denil M, Shakibi B, Dinh L, et al (2013) Predicting parameters in deep learning. ArXiv preprint arXiv:1306.0543. https://proceedings.neurips.cc/paper/2013/hash/7fec306d1e665bc9c748b5d2b99a6e97-Abstract.html

Denton EL, Zaremba W, Bruna J, et al (2014) Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in neural information processing systems, pp 1269–1277. https://proceedings.neurips.cc/paper/2014/hash/2afe4567e1bf64d32a5527244d104cea-Abstract.html

Diethe T, Twomey N, Flach PA (2016) Active transfer learning for activity recognition. In: 24th European symposium on artificial neural networks, ESANN 2016, Bruges, Belgium, April 27–29, 2016. http://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2016-99.pdf

Dosovitskiy A, Beyer L, Kolesnikov A, et al (2020) An image is worth 16x16 words: transformers for image recognition at scale. ArXiv preprint arXiv:2010.11929

Drolia U, Guo K, Narasimhan P (2017a) Precog: prefetching for image recognition applications at the edge. In: Proceedings of the 2nd ACM/IEEE symposium on edge computing, pp 1–13. https://doi.org/10.1145/3132211.3134456

Drolia U, Guo K, Tan J, et al (2017b) Cachier: edge-caching for recognition applications. In: 2017 IEEE 37th international conference on distributed computing systems (ICDCS). IEEE, pp 276–286. https://doi.org/10.1109/ICDCS.2017.94

Du G, Zhang J, Luo Z et al (2020) Joint imbalanced classification and feature selection for hospital readmissions. Knowl-Based Syst 200(106):020. https://doi.org/10.1016/j.knosys.2020.106020

Du G, Zhang J, Ma F et al (2021) Towards graph-based class-imbalance learning for hospital readmission. Expert Syst Appl 176(114):791. https://doi.org/10.1016/j.eswa.2021.114791

Du K, Pervaiz A, Yuan X, et al (2020b) Server-driven video streaming for deep learning inference. In: Proceedings of the annual conference of the ACM special interest group on data communication on the applications, technologies, architectures, and protocols for computer communication. association for computing machinery, New York, NY, USA, pp 557–570. https://doi.org/10.1145/3387514.3405887

Duan M, Liu D, Chen X, et al (2019) Astraea: self-balancing federated learning for improving classification accuracy of mobile deep learning applications. In: 2019 IEEE 37th international conference on computer design (ICCD). IEEE, pp 246–254. https://doi.org/10.1109/ICCD46524.2019.00038

Duan M, Liu D, Chen X et al (2020) Self-balancing federated learning with global imbalanced data in mobile systems. IEEE Trans Parallel Distrib Syst 32(1):59–71. https://doi.org/10.1109/TPDS.2020.3009406

Dwisnanto Putro M, Nguyen DL, Jo KH (2020) Fast eye detector using CPU based lightweight convolutional neural network. In: 2020 20th international conference on control, automation and systems (ICCAS), pp 12–16. https://doi.org/10.23919/ICCAS50221.2020.9268234

Elsken T, Metzen JH, Hutter F (2019) Neural architecture search: a survey. J Mach Learn Res 20(1):1997–2017. http://jmlr.org/papers/v20/18-598.html

Geyer RC, Klein T, Nabi M (2017) Differentially private federated learning: a client level perspective. ArXiv preprint arXiv:1712.07557

Gibiansky A (2017) Bringing HPC techniques to deep learning. Baidu Research, Tech Rep. http://research.baidu.com/bringing-hpc-techniques-deep-learning

Gong Y, Liu L, Yang M, et al (2014) Compressing deep convolutional networks using vector quantization. ArXiv preprint arXiv:1412.6115

Group OCAW, et al (2017) Openfog reference architecture for fog computing. OPFRA001 20817:162. https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf

Guo J, Li Y, Lin W, et al (2018a) Network decoupling: From regular to depthwise separable convolutions. ArXiv preprint arXiv:1808.05517

Guo P, Hu B, Li R, et al (2018b) Foggycache: cross-device approximate computation reuse. In: Proceedings of the 24th annual international conference on mobile computing and networking, pp 19–34. https://doi.org/10.1145/3241539.3241557

Guo Y, Yao A, Chen Y (2016) Dynamic network surgery for efficient DNNs. ArXiv preprint arXiv:1608.04493

Gupta S, Agrawal A, Gopalakrishnan K, et al (2015) Deep learning with limited numerical precision. In: International conference on machine learning. PMLR, pp 1737–1746. http://proceedings.mlr.press/v37/gupta15.html

Han S, Mao H, Dally WJ (2015a) Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. ArXiv preprint arXiv:1510.00149

Han S, Pool J, Tran J, et al (2015b) Learning both weights and connections for efficient neural networks. ArXiv preprint arXiv:1506.02626

Han S, Liu X, Mao H, et al (2016) EIE: efficient inference engine on compressed deep neural network. In: 43rd ACM/IEEE annual international symposium on computer architecture, ISCA 2016, Seoul, South Korea, June 18–22, 2016. IEEE Computer Society, pp 243–254. https://doi.org/10.1109/ISCA.2016.30

Hartmann F, Suh S, Komarzewski A, et al (2019) Federated learning for ranking browser history suggestions. CoRR. http://arxiv.org/abs/1911.11807

Hassibi B, Stork DG, Wolff GJ (1993) Optimal brain surgeon and general network pruning. In: IEEE international conference on neural networks. IEEE, pp 293–299. https://doi.org/10.1109/ICNN.1993.298572

He K, Zhang X, Ren S, et al (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778. https://doi.org/10.1109/CVPR.2016.90

He Y, Zhang X, Sun J (2017) Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE international conference on computer vision, pp 1389–1397. https://doi.org/10.1109/ICCV.2017.155

Hegedűs I, Danner G, Jelasity M (2019) Gossip learning as a decentralized alternative to federated learning. In: IFIP international conference on distributed applications and interoperable systems. Springer, pp 74–90. https://doi.org/10.1007/978-3-030-22496-7_5

Heo B, Lee M, Yun S, et al (2019) Knowledge distillation with adversarial samples supporting decision boundary. In: Proceedings of the AAAI conference on artificial intelligence, pp 3771–3778. http://arxiv.org/abs/1805.05532

Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. ArXiv preprint arXiv:1503.02531. https://doi.org/10.4140/TCP.n.2015.249

Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

Hitaj B, Ateniese G, Perez-Cruz F (2017) Deep models under the GAN: information leakage from collaborative deep learning. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp 603–618. https://doi.org/10.1145/3133956.3134012

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Holi JL, Hwang JN (1993) Finite precision error analysis of neural network hardware implementations. IEEE Trans Comput 42(3):281–290. https://doi.org/10.1109/12.210171

Howard A, Sandler M, Chu G, et al (2019) Searching for mobilenetv3. In: Proceedings of the IEEE/CVF international conference on computer vision, pp 1314–1324. http://arxiv.org/abs/1905.02244

Howard AG, Zhu M, Chen B, et al (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. ArXiv preprint arXiv:1704.04861

Hsieh K, Ananthanarayanan G, Bodik P, et al (2018) Focus: querying large video datasets with low latency and low cost. In: 13th USENIX symposium on operating systems design and implementation (OSDI 18). USENIX Association, Carlsbad, CA, pp 269–286. https://www.usenix.org/conference/osdi18/presentation/hsieh

Hu C, Jiang J, Wang Z (2019) Decentralized federated learning: a segmented gossip approach. ArXiv preprint arXiv:1908.07782

Hu J, Shen L, Sun G (2018a) Squeeze-and-excitation networks. In: 2018 IEEE conference on computer vision and pattern recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018. Computer Vision Foundation/IEEE Computer Society, pp 7132–7141. https://doi.org/10.1109/CVPR.2018.00745

Hu Q, Wang P, Cheng J (2018b) From hashing to CNNs: training binary weight networks via hashing. In: 32nd AAAI conference on artificial intelligence. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16466

Huang G, Sun Y, Liu Z, et al (2016) Deep networks with stochastic depth. In: European conference on computer vision. Springer, pp 646–661. https://doi.org/10.1007/978-3-319-46493-0_39

Huang Y, Wang F, Wang F, et al (2019) Deepar: a hybrid device-edge-cloud execution framework for mobile deep learning applications. In: IEEE INFOCOM 2019-IEEE conference on computer communications workshops (INFOCOM WKSHPS). IEEE, pp 892–897. https://doi.org/10.1109/INFCOMW.2019.8845240

Iandola FN, Han S, Moskewicz MW, et al (2016) Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 MB model size. ArXiv preprint arXiv:1602.07360

Jaderberg M, Vedaldi A, Zisserman A (2014) Speeding up convolutional neural networks with low rank expansions. ArXiv preprint arXiv:1405.3866

Jain S, Zhang X, Zhou Y, et al (2018) Rexcam: Resource-efficient, cross-camera video analytics at scale. ArXiv preprint arXiv:1811.01268

Jain S, Zhang X, Zhou Y, et al (2020) Spatula: efficient cross-camera video analytics on large camera networks. In: 2020 IEEE/ACM symposium on edge computing (SEC). IEEE, pp 110–124. https://doi.org/10.1109/SEC50012.2020.00016

Janjua ZH, Vecchio M, Antonini M et al (2019) IRESE: an intelligent rare-event detection system using unsupervised learning on the IOT edge. Eng Appl Artif Intell 84:41–50. https://doi.org/10.1016/j.engappai.2019.05.011

Jiang Y, Wang S, Valls V, et al (2019) Model pruning enables efficient federated learning on edge devices. ArXiv preprint arXiv:1909.12326

Kang D, Emmons J, Abuzaid F, et al (2017a) Noscope: optimizing neural network queries over video at scale. ArXiv preprint arXiv:1703.02529. https://doi.org/10.14778/3137628.3137664

Kang Y, Hauswald J, Gao C et al (2017) Neurosurgeon: collaborative intelligence between the cloud and mobile edge. ACM SIGARCH Comput Arch News 45(1):615–629. https://doi.org/10.1145/3037697.3037698

Kholod I, Yanaki E, Fomichev D et al (2021) Open-source federated learning frameworks for IOT: a comparative review and analysis. Sensors 21(1):167. https://doi.org/10.3390/s21010167

Kim H, Park J, Bennis M et al (2019) Blockchained on-device federated learning. IEEE Commun Lett 24(6):1279–1283. https://doi.org/10.1109/LCOMM.2019.2921755

Kim J, Park S, Kwak N (2018) Paraphrasing complex network: network compression via factor transfer. ArXiv preprint arXiv:1802.04977

Kim YD, Park E, Yoo S, et al (2015) Compression of deep convolutional neural networks for fast and low power mobile applications. ArXiv preprint arXiv:1511.06530. https://doi.org/10.1109/ICCV.2015.73

Ko JH, Na T, Amir MF, et al (2018) Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In: 2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS). IEEE, pp 1–6. http://arxiv.org/abs/1802.03835

Konečný J, McMahan HB, Yu FX, et al (2016) Federated learning: strategies for improving communication efficiency. ArXiv preprint arXiv:1610.05492

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Adv Neural Inf Process Syst 25:1097–1105. https://doi.org/10.1145/3065386

Lalitha A, Kilinc OC, Javidi T, et al (2019) Peer-to-peer federated learning on graphs. ArXiv preprint arXiv:1901.11173

Lane ND, Bhattacharya S, Georgiev P, et al (2016) Deepx: a software accelerator for low-power deep learning inference on mobile devices. In: 2016 15th ACM/IEEE international conference on

information processing in sensor networks (IPSN). IEEE, pp 1–12. https://doi.org/10.1109/IPSN.2016.7460664
Laskaridis S, Venieris SI, Almeida M, et al (2020) Spinn: synergistic progressive inference of neural networks over device and cloud. In: Proceedings of the 26th annual international conference on mobile computing and networking, pp 1–15. https://doi.org/10.1145/3372224.3419194
Lebedev V, Ganin Y, Rakhuba M, et al (2014) Speeding-up convolutional neural networks using fine-tuned CP-decomposition. ArXiv preprint arXiv:1412.6553
LeCun Y, Boser BE, Denker JS et al (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551. https://doi.org/10.1162/neco.1989.1.4.541
LeCun Y, Denker JS, Solla SA (1990) Optimal brain damage. In: Advances in neural information processing systems, pp 598–605. http://papers.nips.cc/paper/250-optimal-brain-damage
Lee C, Hong S, Hong S et al (2020) Performance analysis of local exit for distributed deep neural networks over cloud and edge computing. ETRI J 42(5):658–668. https://doi.org/10.4218/etrij.2020-0112
Lee R, Venieris SI, Dudziak L, et al (2019) Mobisr: Efficient on-device super-resolution through heterogeneous mobile processors. In: The 25th annual international conference on mobile computing and networking, pp 1–16. https://doi.org/10.1145/3300061.3345455
Lee S, Kim H, Jeong B et al (2021) A training method for low rank convolutional neural networks based on alternating tensor compose-decompose method. Appl Sci 11(2):643. https://doi.org/10.3390/app11020643
Li D, Wang X, Kong D (2018a) Deeprebirth: accelerating deep neural network execution on mobile devices. In: Proceedings of the AAAI conference on artificial intelligence. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16652
Li D, Tasci S, Ghosh S, et al (2019a) RILOD: near real-time incremental learning for object detection at the edge. In: Chen S, Onishi R, Ananthanarayanan G, et al (eds) Proceedings of the 4th ACM/IEEE symposium on edge computing, SEC 2019, Arlington, Virginia, USA, November 7–9, 2019. ACM, pp 113–126. https://doi.org/10.1145/3318216.3363317
Li E, Zeng L, Zhou Z et al (2019) Edge AI: on-demand accelerating deep neural network inference via edge computing. IEEE Trans Wirel Commun 19(1):447–457. https://doi.org/10.1109/TWC.2019.2946140
Li F, Zhang B, Liu B (2016a) Ternary weight networks. ArXiv preprint arXiv:1605.04711
Li H, Kadav A, Durdanovic I, et al (2016b) Pruning filters for efficient convnets. ArXiv preprint arXiv:1608.08710
Li H, Hu C, Jiang J, et al (2018b) Jalad: joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In: 2018 IEEE 24th international conference on parallel and distributed systems (ICPADS). IEEE, pp 671–678. https://doi.org/10.1109/PADSW.2018.8645013
Li L, Ota K, Dong M (2018) Deep learning for smart industry: efficient manufacture inspection system with fog computing. IEEE Trans Industr Inf 14(10):4665–4673. https://doi.org/10.1109/TII.2018.2842821
Li M, Xie L, Lv Z, et al (2020) Multistep deep system for multimodal emotion detection with invalid data in the internet of things. IEEE Access 8:187,208–187,221. https://doi.org/10.1109/ACCESS.2020.3029288
Li X, Huang K, Yang W, et al (2019c) On the convergence of fedavg on non-iid data. ArXiv preprint arXiv:1907.02189
Li Y, Lin S, Zhang B, et al (2019d) Exploiting kernel sparsity and entropy for interpretable CNN compression. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 2800–2809. https://doi.org/10.1109/CVPR.2019.00291
Lin C, Zhong Z, Wu W, et al (2018a) Synaptic strength for convolutional neural network. ArXiv preprint arXiv:1811.02454
Lin M, Chen Q, Yan S (2013) Network in network. ArXiv preprint arXiv:1312.4400
Lin S, Ji R, Chen C et al (2018) Holistic CNN compression via low-rank decomposition with knowledge transfer. IEEE Trans Pattern Anal Mach Intell 41(12):2889–2905. https://doi.org/10.1109/TPAMI.2018.2873305
Lin X, Zhao C, Pan W (2017a) Towards accurate binary convolutional neural network. ArXiv preprint arXiv:1711.11294
Lin Y, Han S, Mao H, et al (2017b) Deep gradient compression: reducing the communication bandwidth for distributed training. ArXiv preprint arXiv:1712.01887
Ling S, Pastor A, Li J, et al (2020) Few-shot pill recognition. In: 2020 IEEE/CVF conference on computer vision and pattern recognition, CVPR 2020, Seattle, WA, USA, June 13–19, 2020. Computer Vision Foundation/IEEE, pp 9786–9795. https://doi.org/10.1109/CVPR42600.2020.00981

Liu L, Li H, Gruteser M (2019) Edge assisted real-time object detection for mobile augmented reality. In: The 25th annual international conference on mobile computing and networking, pp 1–16. https://doi.org/10.1145/3300061.3300116

Liu M, Ding X, Du W (2020) Continuous, real-time object detection on mobile devices without offloading. In: 2020 IEEE 40th international conference on distributed computing systems (ICDCS). IEEE, pp 976–986. https://doi.org/10.1109/ICDCS47774.2020.00085

Liu Y, Garg S, Nie J et al (2021) Deep anomaly detection for time-series data in industrial IOT: a communication-efficient on-device federated learning approach. IEEE Internet Things J 8(8):6348–6358. https://doi.org/10.1109/JIOT.2020.3011726

Liu Z, Li J, Shen Z, et al (2017) Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE international conference on computer vision, pp 2736–2744. https://doi.org/10.1109/ICCV.2017.298

Lo C, Su YY, Lee CY, et al (2017) A dynamic deep neural network design for efficient workload allocation in edge computing. In: 2017 IEEE international conference on computer design (ICCD). IEEE, pp 273–280. https://doi.org/10.1109/ICCD.2017.49

Lu S, Zhang Y, Wang Y (2020) Decentralized federated learning for electronic health records. In: 2020 54th annual conference on information sciences and systems (CISS). IEEE, pp 1–5. https://doi.org/10.1109/CISS48834.2020.1570617414

Lu Y, Huang X, Dai Y et al (2019) Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. IEEE Trans Industr Inf 16(6):4177–4186. https://doi.org/10.1109/TII.2019.2942190

Lungu I, Aimar A, Hu Y et al (2020) Siamese networks for few-shot learning on edge embedded devices. IEEE J Emerg Sel Topics Circuits Syst 10(4):488–497. https://doi.org/10.1109/JETCAS.2020.3033155

Luo JH, Wu J (2020) Autopruner: an end-to-end trainable filter pruning method for efficient deep model inference. Pattern Recogn 107(107):461. https://doi.org/10.1016/j.patcog.2020.107461

Luo JH, Wu J, Lin W (2017) Thinet: a filter level pruning method for deep neural network compression. In: Proceedings of the IEEE international conference on computer vision, pp 5058–5066. https://doi.org/10.1109/ICCV.2017.541

Luo Y, Yu S (2021) AILC: accelerate on-chip incremental learning with compute-in-memory technology. IEEE Trans Comput 70(8):1225–1238. https://doi.org/10.1109/TC.2021.3053199

Ma N, Zhang X, Zheng HT, et al (2018) Shufflenet v2: Practical guidelines for efficient CNN architecture design. In: Proceedings of the European conference on computer vision (ECCV), pp 116–131. https://doi.org/10.1007/978-3-030-01264-9_8

Manessi F, Rozza A, Bianco S, et al (2018) Automated pruning for deep neural network compression. In: 2018 24th international conference on pattern recognition (ICPR). IEEE, pp 657–664. https://doi.org/10.1109/ICPR.2018.8546129

Mao J, Chen X, Nixon KW, et al (2017a) Modnn: Local distributed mobile computing system for deep neural network. In: Design, automation & test in Europe conference & exhibition (DATE). IEEE, pp 1396–1401. https://doi.org/10.23919/DATE.2017.7927211

Mao J, Yang Z, Wen W, et al (2017b) Mednn: a distributed mobile system with enhanced partition and deployment for large-scale DNNs. In: 2017 IEEE/ACM international conference on computer-aided design (ICCAD). IEEE, pp 751–756. https://doi.org/10.1109/ICCAD.2017.8203852

Marco VS, Taylor B, Wang Z, et al (2019) Optimizing deep learning inference on embedded systems through adaptive model selection. CoRR. http://arxiv.org/abs/1911.04946

Martinez B, Yang J, Bulat A, et al (2020) Training binary neural networks with real-to-binary convolutions. ArXiv preprint arXiv:2003.11535

Mathur A, Zhang T, Bhattacharya S, et al (2018) Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices. In: Mottola L, Gao J, Zhang P (eds) Proceedings of the 17th ACM/IEEE international conference on information processing in sensor networks, IPSN 2018, Porto, Portugal, April 11–13, 2018. IEEE/ACM, pp 200–211. https://doi.org/10.1109/IPSN.2018.00048

McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5(4):115–133. https://doi.org/10.1007/BF02459570

McMahan B, Moore E, Ramage D, et al (2017) Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. PMLR, pp 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a.html

Melis L, Song C, De Cristofaro E, et al (2019) Exploiting unintended feature leakage in collaborative learning. In: 2019 IEEE symposium on security and privacy (SP). IEEE, pp 691–706. https://doi.org/10.1109/SP.2019.00029

Mell P, Grance T et al (2011) The NIST definition of cloud computing. https://doi.org/10.6028/NIST.SP.800-145

Mirzadeh SI, Farajtabar M, Li A, et al (2020) Improved knowledge distillation via teacher assistant. In: Proceedings of the AAAI conference on artificial intelligence, pp 5191–5198. https://aaai.org/ojs/index.php/AAAI/article/view/5963

Mishra A, Marr D (2017) Apprentice: using knowledge distillation techniques to improve low-precision network accuracy. ArXiv preprint arXiv:1711.05852

Mishra A, Nurvitadhi E, Cook JJ, et al (2017) WRPN: wide reduced-precision networks. ArXiv preprint arXiv:1709.01134

Mnih V, Kavukcuoglu K, Silver D, et al (2013) Playing atari with deep reinforcement learning. CoRR. http://arxiv.org/abs/1312.5602

Molchanov P, Tyree S, Karras T, et al (2016) Pruning convolutional neural networks for resource efficient inference. ArXiv preprint arXiv:1611.06440. https://openreview.net/forum?id=SJGCiw5gl

Novikov A, Podoprikhin D, Osokin A, et al (2015) Tensorizing neural networks. ArXiv preprint arXiv:1509.06569

Pakha C, Chowdhery A, Jiang J (2018) Reinventing video streaming for distributed vision analytics. In: 10th USENIX workshop on hot topics in cloud computing (HotCloud 18). USENIX Association, Boston. https://www.usenix.org/conference/hotcloud18/presentation/pakha

Panda P, Ankit A, Wijesinghe P, et al (2016) Falcon: feature driven selective classification for energy-efficient image recognition. IEEE Trans Comput-Aided Des Integr Circuits Syst. https://doi.org/10.1109/TCAD.2017.2681075

Panda P, Sengupta A, Roy K (2017) Energy-efficient and improved image recognition with conditional deep learning. ACM J Emerging Technol Comput Syst (JETC) 13(3):1–21. https://doi.org/10.1145/3007192

Park E, Kim D, Kim S, et al (2015) Big/little deep neural network for ultra low power inference. In: 2015 international conference on hardware/software codesign and system synthesis (CODES+ISSS). https://doi.org/10.1109/CODESISSS.2015.7331375

Patarasuk P, Yuan X (2009) Bandwidth optimal all-reduce algorithms for clusters of workstations. J Parallel Distrib Comput 69(2):117–124. https://doi.org/10.1016/j.jpdc.2008.09.002

Qi T, Wu F, Wu C, et al (2020) Privacy-preserving news recommendation model training via federated learning. CoRR. https://arxiv.org/abs/2003.09592

Radu V, Henne M (2019) Vision2sensor: knowledge transfer across sensing modalities for human activity recognition. Proc ACM Interact Mob Wearable Ubiquitous Technol 3(3):84:1–84:21. https://doi.org/10.1145/3351242

Rastegari M, Ordonez V, Redmon J, et al (2016) Xnor-net: imagenet classification using binary convolutional neural networks. In: European conference on computer vision. Springer, pp 525–542. https://doi.org/10.1007/978-3-319-46493-0_32

Reisizadeh A, Mokhtari A, Hassani H, et al (2020) Fedpaq: a communication-efficient federated learning method with periodic averaging and quantization. In: International conference on artificial intelligence and statistics. PMLR, pp 2021–2031. http://proceedings.mlr.press/v108/reisizadeh20a.html

Rigamonti R, Sironi A, Lepetit V, et al (2013) Learning separable filters. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2754–2761. https://doi.org/10.1109/CVPR.2013.355

Romero A, Ballas N, Kahou SE, et al (2014) Fitnets: hints for thin deep nets. ArXiv preprint arXiv:1412.6550

Roy AG, Siddiqui S, Pölsterl S, et al (2019) Braintorrent: a peer-to-peer environment for decentralized federated learning. ArXiv preprint arXiv:1905.06731

Sainath TN, Kingsbury B, Sindhwani V, et al (2013) Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, pp 6655–6659. https://doi.org/10.1109/ICASSP.2013.6638949

Samarakoon S, Bennis M, Saad W et al (2020) Distributed federated learning for ultra-reliable low-latency vehicular communications. IEEE Trans Commun 68(2):1146–1159. https://doi.org/10.1109/TCOMM.2019.2956472

Sanchez-Iborra R, Skarmeta AF (2020) Tinyml-enabled frugal smart objects: Challenges and opportunities. IEEE Circuits Syst Mag 20(3):4–18. https://doi.org/10.1109/MCAS.2020.3005467

Sandler M, Howard A, Zhu M, et al (2018) Mobilenetv2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

Sau BB, Balasubramanian VN (2016) Deep model compression: distilling knowledge from noisy teachers. ArXiv preprint arXiv:1610.09650

Savazzi S, Nicoli M, Rampa V (2020) Federated learning with cooperating devices: a consensus approach for massive IoT networks. IEEE Internet Things J 7(5):4641–4654. https://doi.org/10.1109/JIOT.2020.2964162

Seide F, Fu H, Droppo J, et al (2014) 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In: 15th annual conference of the international speech communication association. http://www.isca-speech.org/archive/interspeech_2014/i14_1058.html

Shahmohammadi F, Hosseini A, King CE, et al (2017) Smartwatch based activity recognition using active learning. In: Bonato P, Wang H (eds) Proceedings of the 2nd IEEE/ACM international conference on connected health: applications, systems and engineering technologies, CHASE 2017, Philadelphia, PA, USA, July 17–19, 2017. IEEE Computer Society/ACM, pp 321–329. https://doi.org/10.1109/CHASE.2017.115

Sheller MJ, Reina GA, Edwards B, et al (2018) Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In: International MICCAI Brainlesion workshop. Springer, pp 92–104. https://doi.org/10.1007/978-3-030-11723-8_9

Shi W, Cao J, Zhang Q et al (2016) Edge computing: vision and challenges. IEEE Internet Things J 3(5):637–646. https://doi.org/10.1109/JIOT.2016.2579198

Shokri R, Shmatikov V (2015) Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, pp 1310–1321. https://doi.org/10.1109/ALLERTON.2015.7447103

Silver D, Huang A, Maddison CJ, et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489. https://doi.org/10.1038/nature16961

Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. ArXiv preprint ArXiv:1409.1556

Smola A, Narayanamurthy S (2010) An architecture for parallel topic models. Proc VLDB Endow 3(1-2):703–710. https://doi.org/10.14778/1920841.1920931

Soudry D, Hubara I, Meir R (2014) Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In: NIPS, p 2. https://proceedings.neurips.cc/paper/2014/hash/076a0c97d09cf1a0ec3e19c7f2529f2b-Abstract.html

Srivastava N, Hinton G, Krizhevsky A, et al (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958. http://dl.acm.org/citation.cfm?id=2670313

Stahl R, Hoffman A, Mueller-Gritschneder D, et al (2021) Deeperthings: fully distributed CNN inference on resource-constrained edge devices. Int J Parallel Progr. https://doi.org/10.1007/s10766-021-00712-3

Stamoulis D, Chin T, Prakash AK, et al (2019) Designing adaptive neural networks for energy-constrained image classification. In: 2018 IEEE/ACM international conference on computer-aided design (ICCAD). https://doi.org/10.1145/3240765.3240796

Swaminathan S, Garg D, Kannan R et al (2020) Sparse low rank factorization for deep neural network compression. Neurocomputing 398:185–196. https://doi.org/10.1016/j.neucom.2020.02.035

Szegedy C, Liu W, Jia Y, et al (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Tan M, Le Q (2019) Efficientnet: rethinking model scaling for convolutional neural networks. In: International conference on machine learning. PMLR, pp 6105–6114. http://proceedings.mlr.press/v97/tan19a.html

Tan M, Le QV (2021) Efficientnetv2: smaller models and faster training. ArXiv preprint arXiv:2104.00298

Tang Z, Shi S, Chu X, et al (2020) Communication-efficient distributed deep learning: a comprehensive survey. CoRR. https://arxiv.org/abs/2003.06307

Tann H, Hashemi S, Bahar RI et al (2016) Runtime configurable deep neural networks for energy-accuracy trade-off. ACM. https://doi.org/10.1145/2968456.2968458

Taylor B, Marco VS, Wolff W et al (2018) Adaptive deep learning model selection on embedded systems. ACM SIGPLAN Notices 53(6):31–43. https://doi.org/10.1145/3299710.3211336

Teerapittayanon S, McDanel B, Kung HT (2016) Branchynet: fast inference via early exiting from deep neural networks. In: 2016 23rd international conference on pattern recognition (ICPR). IEEE, pp 2464–2469. https://doi.org/10.1109/ICPR.2016.7900006

Tian X, Zhu J, Xu T et al (2021) Mobility-included DNN partition offloading from mobile devices to edge clouds. Sensors 21(1):229. https://doi.org/10.3390/s21010229

Touvron H, Cord M, Douze M, et al (2020) Training data-efficient image transformers & distillation through attention. ArXiv preprint arXiv:2012.12877

Truex S, Baracaldo N, Anwar A, et al (2019) A hybrid approach to privacy-preserving federated learning. In: Proceedings of the 12th ACM workshop on artificial intelligence and security, pp 1–11. https://doi.org/10.1145/3338501.3357370

Vaswani A, Shazeer N, Parmar N, et al (2017) Attention is all you need. In: Advances in neural information processing systems, pp 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Wang J, Feng Z, Chen Z, et al (2018a) Bandwidth-efficient live video analytics for drones via edge computing. In: 2018 IEEE/ACM symposium on edge computing (SEC). IEEE, pp 159–173. https://doi.org/10.1109/SEC.2018.00019

Wang P, Cheng J (2016) Accelerating convolutional neural networks for mobile applications. In: Proceedings of the 24th ACM international conference on multimedia, pp 541–545. https://doi.org/10.1145/2964284.2967280

Wang S, Tuor T, Salonidis T et al (2019) Adaptive federated learning in resource constrained edge computing systems. IEEE J Sel Areas Commun 37(6):1205–1221. https://doi.org/10.1109/JSAC.2019.2904348

Wang X, Yu F, Dou ZY, et al (2018b) Skipnet: learning dynamic routing in convolutional networks. In: Proceedings of the European conference on computer vision (ECCV), pp 409–424, https://doi.org/10.1007/978-3-030-01261-8_25

Wang X, Han Y, Leung V et al (2020) Convergence of edge computing and deep learning: a comprehensive survey. IEEE Commun Surv Tutor 22(99):869–904. https://doi.org/10.1109/COMST.2020.2970550

Wang X, Yang Z, Wu J, et al (2021) Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision. In: IEEE INFOCOM 2021—IEEE conference on computer communications, pp 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488843

Wei K, Li J, Ding M et al (2020) Federated learning with differential privacy: algorithms and performance analysis. IEEE Trans Inf Forensics Secur 15:3454–3469. https://doi.org/10.1109/TIFS.2020.2988575

Wen W, Wu C, Wang Y, et al (2016) Learning structured sparsity in deep neural networks. Adv Neural Inf Process Syst 29:2074–2082. https://proceedings.neurips.cc/paper/2016/file/41bfd20a38bb1b0bec75acf0845530a7-Paper.pdf

Weng J, Weng J, Zhang J et al (2021) Deepchain: auditable and privacy-preserving deep learning with blockchain-based incentive. IEEE Trans Dependable Secur Comput 18(5):2438–2455. https://doi.org/10.1109/TDSC.2019.2952332

Wistuba M, Rawat A, Pedapati T (2019) A survey on neural architecture search. ArXiv preprint arXiv:1905.01392

Wu J, Leng C, Wang Y, et al (2016) Quantized convolutional neural networks for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4820–4828. https://doi.org/10.1109/CVPR.2016.521

Xie S, Girshick R, Dollár P, et al (2017) Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1492–1500. https://doi.org/10.1109/CVPR.2017.634

Xu D, Li T, Li Y, et al (2020) Edge intelligence: architectures, challenges, and applications. ArXiv preprint arXiv:2003.12172

Xu M, Zhu M, Liu Y, et al (2018) Deepcache: principled cache for mobile deep vision. In: Proceedings of the 24th annual international conference on mobile computing and networking, pp 129–144. https://doi.org/10.1145/3241539.3241563

Xue J, Li J, Gong Y (2013) Restructuring of deep neural network acoustic models with singular value decomposition. In: Interspeech, pp 2365–2369. http://www.isca-speech.org/archive/interspeech_2013/i13_2365.html

Yang L, Chen X, Perlaza SM et al (2020) Special issue on artificial-intelligence-powered edge computing for internet of things. IEEE Internet Things J 7(10):9224–9226. https://doi.org/10.1109/JIOT.2020.3019948

Yang L, Han Y, Chen X, et al (2020b) Resolution adaptive networks for efficient inference. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 2369–2378. https://doi.org/10.1109/CVPR42600.2020.00244

Yang Q, Liu Y, Chen T et al (2019) Federated machine learning: concept and applications. ACM Trans Intell Syst Technol 10(2):1–19. https://doi.org/10.1145/3298981

Yang TJ, Chen YH, Sze V (2017) Designing energy-efficient convolutional neural networks using energy-aware pruning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5687–5695. https://doi.org/10.1109/CVPR.2017.643

Yi J, Choi S, Lee Y (2020) Eagleeye: wearable camera-based person identification in crowded urban spaces. In: MobiCom '20: The 26th annual international conference on mobile computing and networking, London, United Kingdom, September 21–25, 2020. ACM, pp 4:1–4:14. https://doi.org/10.1145/3372224.3380881

Yim J, Joo D, Bae J, et al (2017) A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4133–4141. https://doi.org/10.1109/CVPR.2017.754

Yoshida N, Nishio T, Morikura M, et al (2020) Hybrid-fl for wireless networks: cooperative learning mechanism using non-IID data. In: ICC 2020-2020 IEEE international conference on communications (ICC). IEEE, pp 1–7. https://doi.org/10.1109/ICC40277.2020.9149323

You Z, Yan K, Ye J, et al (2019) Gate decorator: global filter pruning method for accelerating deep convolutional neural networks. ArXiv preprint arXiv:1909.08174

Yu R, Li A, Chen CF, et al (2018) Nisp: pruning networks using neuron importance score propagation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 9194–9203. https://doi.org/10.1109/CVPR.2018.00958

Zagoruyko S, Komodakis N (2016a) Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer. ArXiv preprint arXiv:1612.03928

Zagoruyko S, Komodakis N (2016b) Wide residual networks. ArXiv preprint arXiv:1605.07146

Zeng L, Li E, Zhou Z et al (2019) Boomerang: on-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things. IEEE Netw 33(5):96–103. https://doi.org/10.1109/MNET.001.1800506

Zeng X, Cao K, Zhang M (2017) *MobileDeepPill*: a small-footprint mobile deep learning system for recognizing unconstrained pill images. In: Choudhury T, Ko SY, Campbell A, et al (eds) Proceedings of the 15th annual international conference on mobile systems, applications, and services, MobiSys'17, Niagara Falls, NY, USA, June 19–23, 2017. ACM, pp 56–67. https://doi.org/10.1145/3081333.3081336

Zhang C, Cao Q, Jiang H, et al (2018a) Ffs-va: a fast filtering system for large-scale video analytics. In: Proceedings of the 47th international conference on parallel processing, pp 1–10. https://doi.org/10.1145/3225058.3225103

Zhang C, Cao Q, Jiang H et al (2020) A fast filtering mechanism to improve efficiency of large-scale video analytics. IEEE Trans Comput 69(6):914–928. https://doi.org/10.1109/TC.2020.2970413

Zhang L, Song J, Gao A, et al (2019) Be your own teacher: improve the performance of convolutional neural networks via self distillation. In: Proceedings of the IEEE/CVF international conference on computer vision, pp 3713–3722. https://doi.org/10.1109/ICCV.2019.00381

Zhang W, Li X, Ma H et al (2021) Federated learning for machinery fault diagnosis with dynamic validation and self-supervision. Knowl Based Syst 213(106):679. https://doi.org/10.1016/j.knosys.2020.106679

Zhang W, Wang X, Zhou P, et al (2021b) Client selection for federated learning with non-IID data in mobile edge computing. IEEE Access 9:24,462–24,474. https://doi.org/10.1109/ACCESS.2021.3056919

Zhang X, Zou J, He K et al (2015) Accelerating very deep convolutional networks for classification and detection. IEEE Trans Pattern Anal Mach Intell 38(10):1943–1955. https://doi.org/10.1109/TPAMI.2015.2502579

Zhang X, Zhou X, Lin M, et al (2018b) Shufflenet: an extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6848–6856. https://doi.org/10.1109/CVPR.2018.00716

Zhang Y, Wallace B (2015) A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. ArXiv preprint arXiv:1510.03820

Zhang Y, Xiang T, Hospedales TM, et al (2018c) Deep mutual learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4320–4328. http://arxiv.org/abs/1706.00384

Zhao Y, Li M, Lai L, et al (2018a) Federated learning with non-IID data. ArXiv preprint arXiv:1806.00582

Zhao Y, Zhao J, Jiang L et al (2020) Privacy-preserving blockchain-based federated learning for IoT devices. IEEE Internet Things J 8(3):1817–1829. https://doi.org/10.1109/JIOT.2020.3017377

Zhao Z, Barijough KM, Gerstlauer A (2018) Deepthings: distributed adaptive deep learning inference on resource-constrained IoT edge clusters. IEEE Trans Comput Aided Des Integr Circuits Syst 37(11):2348–2359. https://doi.org/10.1109/TCAD.2018.2858384

Zhou A, Yao A, Guo Y, et al (2017) Incremental network quantization: towards lossless CNNs with low-precision weights. ArXiv preprint arXiv:1702.03044

Zhou G, Fan Y, Cui R, et al (2018) Rocket launching: a universal and efficient framework for training well-performing light net. In: 32nd AAAI conference on artificial intelligence. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16090

Zhou J, Wang Y, Ota K et al (2019) Aaiot: accelerating artificial intelligence in IoT systems. IEEE Wirel Commun Lett 8(3):825–828. https://doi.org/10.1109/LWC.2019.2894703

Zhou S, Wu Y, Ni Z, et al (2016) Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients. ArXiv preprint arXiv:1606.06160

Zhou Z, Chen X, Li E et al (2019) Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proc IEEE 107(8):1738–1762. https://doi.org/10.1109/JPROC.2019.2918951

Zhu J, Zhao Y, Pei J (2021) Progressive kernel pruning based on the information mapping sparse index for CNN compression. IEEE Access 9:10,974–10,987. https://doi.org/10.1109/ACCESS.2021.3051504

Zuo Y, Chen B, Shi T, et al (2020) Filter pruning without damaging networks capacity. IEEE Access 8:90,924–90,930. https://doi.org/10.1109/ACCESS.2020.2993932