Chapter 9
Intelligence Inference on IoT Devices

Qiyang Zhang, Ying Li, Dingge Zhang, Ilir Murturi, Victor Casamayor Pujol, Schahram Dustdar, and Shangguang Wang

9.1 Introduction

IoT devices (including smartphones and smart tablets) have gained significant popularity and have become the primary gateway to the Internet (Xu et al. 2019, 2020). Meanwhile, the exceptional performance of deep learning (DL) models in computer vision over the past decade has led to an increased reliance on deep neural networks (DNNs) for cloud-based visual analyses. These DNNs are utilized for diverse tasks such as inference and prediction after deployment. This integration of DNNs and cloud-based visual analyses has facilitated the realization of various applications, including object detection (Girshick et al. 2015), vehicle and person reidentification (Liu et al. 2016), pedestrian detection (Sun et al. 2014), and landmark retrieval (Wang et al. 2017), etc.

Q. Zhang (✉)
State Key Laboratory of Network and Switching, Beijing University of Posts and Telecommunications, Beijing, China

Distributed Systems Group, TU Wien, Vienna, Austria
e-mail: qyzhang@bupt.edu.cn

Y. Li
College of Computer Science and Engineering, Northeastern University, Shenyang, China

Distributed Systems Group, TU Wien, Vienna, Austria
e-mail: liying1771@163.com

D. Zhang · S. Wang
State Key Laboratory of Network and Switching, Beijing University of Posts and Telecommunications, Beijing, China
e-mail: zdg@bupt.edu.cn; sgwang@bupt.edu.cn

I. Murturi · V. C. Pujol · S. Dustdar
Distributed Systems Group, TU Wien, Vienna, Austria
e-mail: imurturi@dsg.tuwien.ac.at; v.casamayor@dsg.tuwien.ac.at; dustdar@dsg.tuwien.ac.at

P. K. Donta et al. (eds.), *Learning Techniques for the Internet of Things*,
https://doi.org/10.1007/978-3-031-50514-0_9

Developers are actively exploring the integration of DL into mobile applications to enhance intelligence and improve the user experience on IoT devices. While DL on the cloud has received significant research attention, the study of DL on IoT devices remains relatively limited. There is a lack of comprehensive understanding regarding the key challenges and issues associated with DL on IoT devices. Therefore, further research is needed to gain insights into this domain.

Supporting a diverse range of devices while maintaining high performance with a single convolutional neural network (CNN) presents a significant challenge. However, the development of models specifically designed for IoT devices has greatly improved the capabilities of AI. These models are optimized for performance and efficiency, considering the limited computational resources, power constraints, and memory limitations of IoT devices. These models are typically lightweight and compact, enabling fast and efficient inference without sacrificing accuracy. They incorporate techniques such as model compression (Choudhary et al. 2020), quantization (Polino et al. 2018), and efficient network architectures (Iandola et al. 2016; Zhang et al. 2018) to minimize computational and memory requirements while achieving high performance.

Device vendors have responded to the demand for efficient CNNs on IoT devices by introducing System-on-Chips (SoCs) and inference libraries that incorporate specialized units for CNN acceleration. These SoCs are equipped with high-performance CPU/GPU units, and dedicated accelerators designed for machine learning (ML) and image processing tasks. While these accelerators enable on-device processing, developers still face the challenge of supporting the diverse array of devices available in the market. The advancements in hardware have significantly enhanced the overall performance and capabilities of IoT devices, allowing them to handle computationally intensive tasks and deliver enhanced user experiences. Moreover, software solutions play a crucial role in accelerating on-device DL inference alongside hardware advancements. For instance, fine-tuned implementation can achieve up to a 62,806$\times$ performance improvement compared to vanilla implementations (Leiserson et al. 2020; Zhang et al. 2022). Inference libraries provide developers with the necessary tools and runtime environments to optimize inference on resource-constrained devices. These libraries enable real-time and on-device inference for a wide range of applications, further enhancing the efficiency and effectiveness of DL on IoT devices.

Recently, there has been a notable increase in the adoption of cloud-based visual analysis, driven by advancements in network infrastructure. To achieve SOTA performance and ensure compatibility with a wide range of IoT devices, developers often choose to offload computational tasks, either partially or entirely, to high-computing-power infrastructures such as cloud servers. The rapid development of 5G communication technology has further facilitated offloading, allowing applications with stringent latency requirements like to be supported effectively. While offloading offers benefits such as improved inference latency and the ability to handle device diversity, it also comes with certain challenges. One of these challenges is the high operational costs associated with maintaining and utilizing cloud resources. Additionally, remote execution raises concerns regarding privacy and security, and the user experience can be affected by variations in networking conditions. To address these challenges, researchers have been exploring collaborative approaches that leverage both local and cloud resources for CNN inference (Huang et al. 2020; Laskaridis et al. 2020). These approaches aim to strike a balance between leveraging the computing power of cloud and utilizing local resources to enhance performance and reduce latency. By distributing the computational workload and optimizing resource utilization, these collaborative methods offer potential solutions to the limitations of cloud-based visual analysis, paving the way for more efficient and effective AI applications.

In summary, IoT devices have made computing pervasive, accessible, and personalized, enriching our daily lives and opening up new possibilities for applications and services in various domains. The remainder of this work is structured as follows: Sect. 9.2 introduces the preliminary work on inference. Section 9.3 explores the diverse applications of inference in IoT, highlighting the range of domains where inference finds utility. Sections 9.4–9.6 present comprehensive reviews of commodity hardware, model optimization, and inference libraries, focusing on their relevance and effectiveness in IoTs, respectively. Section 9.7 reviews the current inference system in edge computing. Section 9.8 presents the research challenges and future opportunities. Lastly, Sect. 9.9 concludes the paper.

9.2 Inference on IoT Devices: Preliminaries

DL model deployment involves two main stages: model training and inference (Xu et al. 2022; Wang et al. 2022; Li et al. 2023). During the training stage, a significant volume of training data is utilized, and the backpropagation algorithm is employed to determine the optimal model parameter values. This process necessitates substantial computing resources and is typically conducted offline. On the other hand, model inference involves utilizing a trained model to process individual or continuous input data. The results of these computations often require real-time feedback to users, making factors such as computing time and system overhead (e.g., memory usage, energy consumption) crucial considerations. This two-stage deployment methodology allows for efficient utilization of computing resources during the training stage and facilitates real-time inference on IoT devices.

Inference refers to the execution of data analysis, decision-making procedures, and related tasks directly on edge devices or servers situated within a decentralized computing infrastructure, thus mitigating the exclusive dependence on cloud-based computing systems. This strategy facilitates timely, context-aware decision-making processes near the network edge, in closer proximity to the data source, proffering numerous advantages and opportunities for IoT devices:

- Real time: Inference empowers devices to make immediate decisions and take action without relying on cloud connectivity. By processing data locally, proximate to the data source, devices can provide real-time responsiveness.

- Reduced Bandwidth: Inference enables IoT devices to locally process large data volumes, thereby reducing latency and conserving bandwidth, proving advantageous in situations with limited network connectivity.
- Privacy Enhancement: Inference bolsters data privacy by limiting sensitive data transmission to the cloud. Conducted locally, it ensures sensitive information remains confined to IoT devices, minimizing potential risks and reinforcing privacy.
- Context-aware Decision-making: Inference utilizes contextual data from IoT devices, such as sensor readings and device-specific information, to enhance result accuracy and relevance. This facilitates intelligent, environment-specific decisions, leading to heightened operational efficiency and effectiveness.

Overall, inference underpins the autonomy, responsiveness, and the capacity to handle intricate tasks of IoT devices. It facilitates the evolution and potentiality of the IoT ecosystem, endowing devices with the ability to exploit their computational prowess and make judicious decisions.

9.3 Promising Intelligence Applications

AI applications, due to their complexity and high computational requirements, are housed in cloud centers. However, this computing paradigm struggles to deliver real-time services like analytics and smart manufacturing. Therefore, situating AI applications on IoT devices widens the application scope of AI models. As shown in Fig. 9.1, DL models can execute on edge devices (i.e., IoT devices and edge servers) or depend on cloud centers. In this section, we spotlight several notable AI applications and their merits.

9.3.1 Real-Time Video Analytic

Video analytics, integral to VR/AR, necessitates considerable computational power and extensive storage resources (Xu et al. 2021). Performing such tasks in the cloud often leads to unexpected latency and high bandwidth usage. However, the progression of edge computing allows for the migration of video analytics closer to the data source, mitigating these issues (Dustdar & Murturi 2021).

Video analysis applications, such as face recognition and object detection, benefit from various effective DL algorithms, including artificial neural networks and histogram analysis (Bajrami et al. 2018). Nonetheless, utilizing a singular model for analysis without noise reduction and feature extraction proves challenging. Thus, integrating multiple models often results in enhanced video analytic performance. For instance, face recognition entails several steps, each addressed by a different

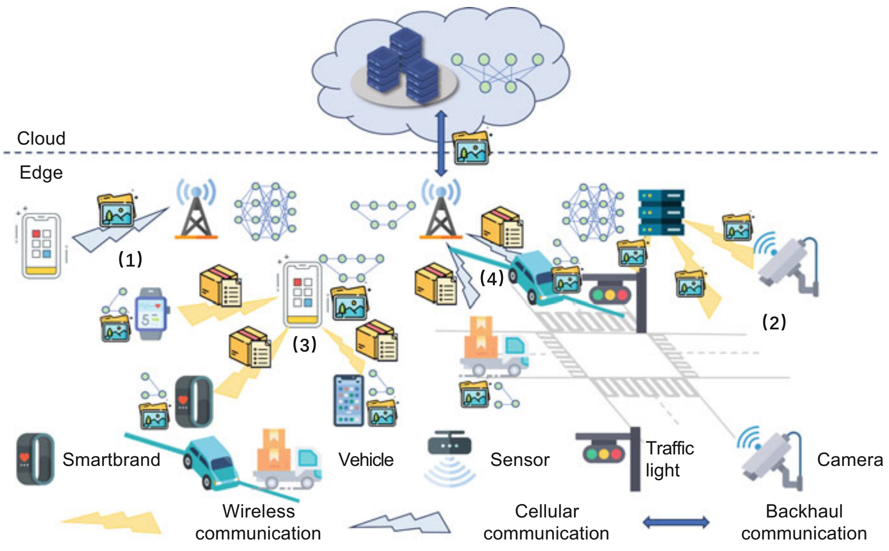

Fig. 9.1 Deep learning models can execute on edge devices (i.e., IoT devices and edge servers) or depend on cloud centers

model: AdaBoost (Viola & Jones 2001) for face detection, a nonlinear SVM classifier for gender and age classification, and a basic algorithm (Lucas & Kanade 1981; Zhou et al. 2022) for face tracking to calculate optimal flow and depict pixel trajectories.

9.3.2 Autonomous Driving

Autonomous vehicles, equipped with a plethora of sensors, generate a vast amount of data necessitating swift processing. The interconnectivity of these vehicles enhances safety, streamlines efficiency, and mitigates traffic congestion. Notably, autonomous driving aims to deliver services characterized by low latency, high-speed communication, and rapid response. ML- and DL-based solutions present potential for optimizing the complex operations intrinsic to autonomous vehicles. For example, ML algorithms deployed in self-driving vehicles extract features from raw data to discern real-time road conditions, facilitating informed decision-making. Similarly, for demanding tasks in autonomous driving—such as sensing, perception, and decision-making—DL algorithms process raw data through sensing to reach final decisions (Liu et al. 2019).

9.3.3 Smart Manufacturing

Smart manufacturing fundamentally hinges on automation and data analysis—the former being the primary goal, and the latter serving as an invaluable tool (Li et al. 2018). Ensuring low latency, privacy protection, and risk control is paramount to adhering to these principles. Within the realm of a smart factory, intelligent inference proves beneficial, bolstering computational resources and facilitating resource scheduling and data processing throughout the manufacturing process. Given the exponential proliferation, remote management of DL models and their continuous evaluation have emerged as pressing imperatives. To tackle these challenges, (Soto et al. 2016) pave the way for the development of real-time applications. Furthermore, DL algorithms are set to be significant catalysts propelling the industry's advancement by transforming all stages of the product lifecycle—from design and manufacturing to service—thereby driving substantial productivity enhancements.

9.3.4 Smart City and Home

The proliferation of IoT devices has sparked the emergence of intelligent services in various aspects of home lifestyles, encompassing appliances like smart TV and air conditioners (Kounoudes et al. 2021; Ain et al. 2018). Furthermore, the deployment of multiple IoT sensors and controllers in smart homes has become a prerequisite. Edge computing-based inference assumes a crucial role in optimizing indoor systems, aiming for low latency and high accuracy, thereby enhancing the capabilities and diversity of services. Moreover, extending edge computing beyond individual homes to encompass communities or cities holds significant potential. The inherent characteristic of geographically distributed data sources in urban environments enables location awareness, latency-sensitive monitoring, and intelligent control. For instance, we integrate large-scale ML algorithms, such as data mining combined with semantic learning, to extract advanced insights and patterns from the voluminous data generated by smart homes and cities (Mohammadi & Al-Fuqaha 2018).

9.4 Commodity Hardware for IoT Devices

With advancements in hardware, low-power IoT devices have the capability to independently handle AI tasks without relying on cloud communication. For instance, commodity CPUs, which are widely available in these devices, serve as the primary hardware for executing inference. CPUs play a crucial role in the inference, supported by toolchains and software libraries that facilitate practical inference. These CPUs share similar microarchitectures, allowing for the effective utilization of optimization techniques. However, performing computationally intensive tasks still poses challenges. For instance, processing a single image using the common VGG model (Sengupta et al. 2019), which consists of 13 CNN layers and 3 fully connected neural network (FCNN) layers, may take hundreds of seconds on devices like the Samsung Galaxy S7 (Xiang & Kim 2019).

Mobile GPUs have revolutionized high-dimensional matrix operations, including matrix decomposition and multiplications in CNN (Owens et al. 2008). Notably, GPUs have emerged as a standout option for edge computing, as they consume less power compared to traditional desktop and server GPUs. In particular, the Jetson family of GPUs, including the latest Jetson Nano, showcases a 128-core affordable GPU module that NVIDIA has successfully introduced. Additionally, the concept of caching computation results in CNN has sparked optimizations in frameworks like DeepMon (Huynh et al. 2017). DeepMon implements a range of optimizations specifically designed for processing convolution layers on mobile GPUs, resulting in significantly reduced inference time.

Due to power and cost constraints on devices, traditional CPU- and GPU-based solutions are not always viable. Moreover, devices often need to handle multiple application requests simultaneously, making the use of CPU- and GPU-based solutions impractical. As a result, hardware integrated with FPGA has gained attention for Edge AI applications. FPGA-based solutions offer several advantages in terms of latency and energy efficiency compared to CPUs and GPUs. However, one challenge is that developing efficient algorithms for FPGA is unfamiliar to most programmers, as it requires the transplantation of models programmed for GPUs into the FPGA platform.

There are also AI accelerators specifically designed for inference that have been introduced by several manufacturers. One notable example is the Myriad VPU (Leon et al. 2022), developed by Movidius, which is optimized for computer vision tasks. It can be easily integrated with devices like Raspberry Pi to perform inference. However, these AI accelerators are not widely available on all devices, limiting their accessibility. Additionally, the ecosystem surrounding these accelerators is still in its early stages and tends to be closed due to their black box structure and proprietary inference frameworks. This creates barriers for widespread adoption and usage. For instance, the Edge TPU, currently found only in Google Pixel smartphones, is limited to running models built with TensorFlow (Developers 2022).

Looking ahead, AI accelerators are expected to play a crucial role in IoT devices. With the introduction of powerful AI SoCs, there is potential for significant improvements in inference performance. As hardware accelerators and software frameworks continue to evolve and upgrade, more AI applications will be able to execute directly on IoT devices.

9.5 Model Optimization for IoT Devices

The limited computing resources on IoT devices necessitate developers to make trade-offs between model accuracy and real-time performance requirements, leading to the inability to deploy SOTA models. A fundamental challenge of this trend is the constrained resources of devices. Therefore, performance optimization has been a primary research direction for both academia and industry.

9.5.1 Lightweight Model Design

To optimize the computational overhead of DL inference, one approach is to ensure the lightweight nature of the DL models themselves. This can be achieved through the design of a lightweight model or the compression of a trained model. For example, SqueezeNet demonstrates such optimization by achieving comparable accuracy to AlexNet while utilizing only 2% of the parameters (Iandola et al. 2016).

The key innovation of SqueezeNet lies in its novel convolution method and the introduction of a fire module. As shown in Fig. 9.2, the fire module consists of a squeeze layer and an expand layer. The squeeze layer employs a 1×1 convolution kernel to alter the number of channels while maintaining the resolution (H$\times$W) of the feature map to achieve compression. The subsequent expand layer utilizes 1×1 and 3×3 convolutional layers, whose outputs are combined to obtain the fire module's output. SqueezeNet follows a similar network design concept as VGG, utilizing stacked convolutional operations, with the difference being the incorporation of the fire module. Furthermore, ShuffleNet integrates group convolution to significantly decrease the number of parameters and computational complexity (Zhang et al. 2018). Group convolution divides channels into subgroups, where the output of each subgroup depends solely on the corresponding input subgroup. To address potential issues caused by group convolution, ShuffleNet (Zhang et al. 2018) introduces the shuffle operation, which rearranges the channels within each part to create a new feature map. The architecture of ShuffleNet is inspired by ResNet (Targ et al. 2016), transitioning from the basic ResNet bottleneck unit to the ShuffleNet bottleneck unit and stacking multiple ShuffleNet bottleneck units to form the complete model.

9.5.2 Model Pruning

Numerous researchers have explored techniques to reduce model complexity in DL models through parameter sharing and pruning. In many neural networks, the computationally intensive matrix multiplication in fully connected layers leads to a large number of model parameters and computing. To overcome this challenge, circulant

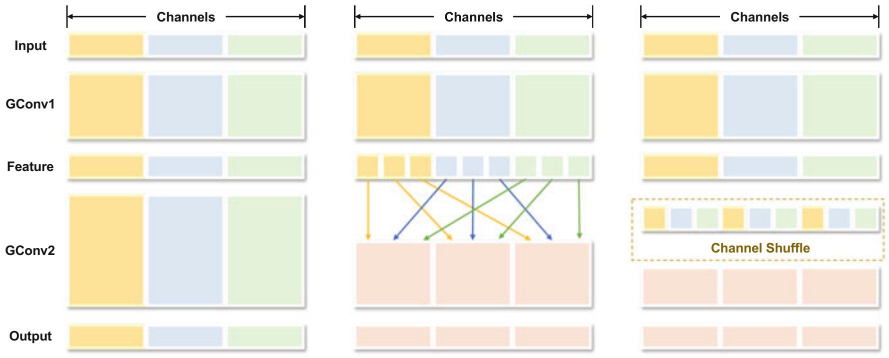

Fig. 9.2 Grouping convolution used in the lightweight model ShuffleNet

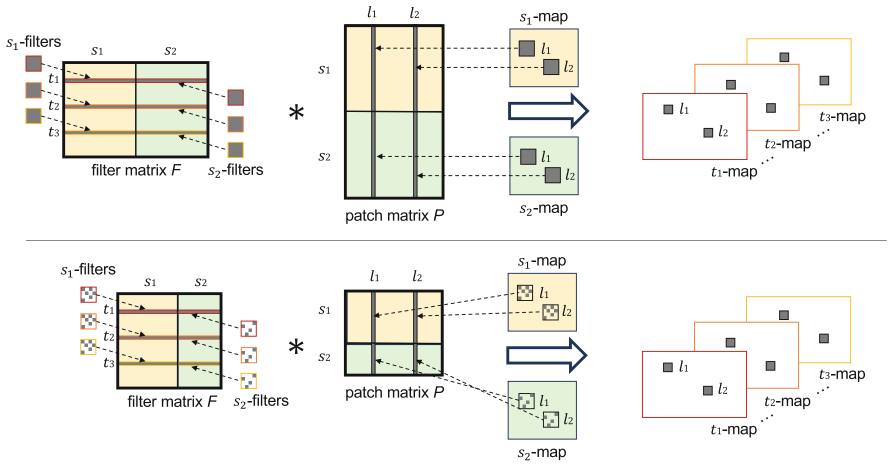

Fig. 9.3 Fast ConvNets reduce operations by pruning convolution kernels

projections have been proposed as a method to accelerate the computation of fully connected layers. As shown in Fig. 9.3, by employing a weight matrix with circulant projections (Cheng et al. 2015), memory requirement is reduced from $O(d^2)$ to $O(d)$ for a matrix of size d × d. Furthermore, the multiplication of rotation matrices can be accelerated using fast Fourier transform (FFT). This technique reduces the computational complexity from $O(d^2)$ to $O(dlogd)$ for the multiplication of a 1 × d vector and a d × d matrix. Given the significant role of CNN models in mobile vision, various approaches have been proposed for parameter pruning and sharing algorithms specifically tailored to CNNs. Fast ConvNets achieves computational reduction by pruning convolution kernels (Lebedev & Lempitsky 2016).

The commonly used technique for implementing convolution operations in DL libraries such as *TensorFlow* and *Caffe* is referred to as *im2col*. This process involves three steps: (1) During convolution, the input image is transformed into a matrix by sliding the convolution kernel. Each column in the matrix represents the information of a small window processed by the kernel. Rows in the matrix correspond to the product of the kernel's height, width, and number of input channels, while the column represents the product of the height and width of the single-channel image output by the convolution layer, representing the overall processing of the small window. (2) By reshaping the convolution kernel, a matrix is obtained where the rows correspond to the number of output image channels and the columns match the row values of the previous matrix. (3) The *im2col* operation converts complex convolution operations into matrix multiplications. The process involves performing matrix multiplication between two matrices and reshaping the resulting matrix into the final output. By leveraging existing optimization algorithms and libraries for efficient matrix operations, such as the BLAS algebraic operation library, *im2col* benefits from optimized matrix operations. Techniques like parameter pruning in Fast ConvNets (Lebedev & Lempitsky 2016) further reduce the matrix dimension after expansion, leading to accelerated computational workload for matrix multiplication.

9.5.3 Model Quantization

Quantization is a technique used to compress DL models by reducing the number of bits required for each parameter. In traditional DL models, parameters are typically represented using 16-bit floating-point numbers. However, experimental studies have shown that using lower-precision representations can significantly reduce memory consumption and computation time without compromising precision. In some cases, researchers have even employed 1-bit representations for storing parameters during both training and inference, achieving comparable results by using values of 1 or -1 (Rastegari et al. 2016). Additionally, other studies have investigated the use of vector quantization and product quantization techniques to compress models and further improve their efficiency.

Vector quantization and product quantization are widely used data compression techniques that involve grouping scalar data into vectors and quantizing them as a whole in the vector space, resulting in lossless data compression. By applying product quantization to the connection layer and the convolution layer, it is possible to achieve benefits such as reduced model size and improved operation time. These techniques are effective in optimizing DL models for improved efficiency and performance (Wu et al. 2016). As illustrated in Fig. 9.4, the main concept involves partitioning the input space into M equally sized subspaces, where each subspace of the weight matrix is assigned a sub-codebook obtained through a clustering algorithm. A codebook consists of multiple codewords, and the core idea of the algorithm is to approximate all subvectors of the same dimension in the space using a limited number of codewords. During inference, the input vector is divided into M sub-vectors, which are then multiplied only with the codewords in their respective subspaces. The final output can then be obtained based on the index of the pre-

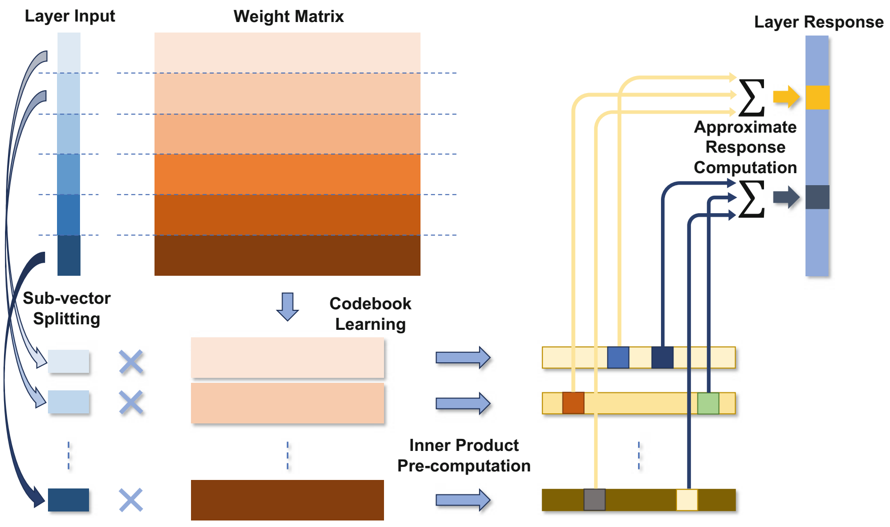

Fig. 9.4 Model compression method based on product quantization

calculated codebook mapped to the output matrix. Consequently, the computational complexity is reduced from the original $O(C_s C_t)$ to $O(C_s K + C_t M)$, where C_s and C_t denote the input and output dimensions, respectively. K represents the number of codewords in each codebook. Experimental results demonstrate that the algorithm can achieve a 4–6× increase in computational speed and reduce the model size by 15–20× with only a marginal 1% loss of accuracy.

9.5.4 Knowledge Distillation

Knowledge distillation (KD) is an effective algorithm widely used for compressing DL models. When employing small models for classification inference, relying solely on one-hot encoding in the training set is insufficient. This encoding method treats categories as independent entities and fails to capture the relationships between them. However, by allowing the small model to learn from the probability distribution generated by a larger model, additional supervision signals are provided, and similarity information between categories is incorporated, facilitating easier learning. For example, in the recognition of handwritten digits, certain images labeled as 3 may bear a resemblance to 8 or 2. One-hot encoding is unable to capture such nuances, whereas a pre-trained large model can provide this information. As a result, researchers modify the loss function to align the small model with the probability distribution outputted by the large model, a process known as KD training.

The effectiveness of KD training has been demonstrated in two datasets: handwriting recognition and speech recognition. However, (Romero et al. 2014) argue that directly mimicking the outputs of large models poses challenges for small models. Additionally, as the model depth increases, emulating the large model becomes more difficult as the supervision signal from the final layer needs to propagate to the earlier layers. To address this challenge, the researchers propose Fitnets, which involve incorporating supervisory signals in the middle layers. By comparing and minimizing the discrepancy between the outputs of the intermediate layers in both the large and small models, the small model can learn from the larger model during an intermediate step of prediction. Here, the "small" refers to the width of the layers rather than the depth. This training approach, known as hint training, involves pre-training the parameters of the first half of the small model using hint training. Subsequently, KD training is employed to train all parameters, enabling the small model to better emulate the knowledge of each layer in the larger model. However, it is important to note that this more active learning method may not be universally applicable due to the significant capacity gap between large and small models.

Building upon the work of (Yim et al. 2017), researchers have extended the concepts and applications of KD. Instead of having the small model directly fit the output of the large model, the focus is on aligning the relationships between the layers of the two models. These relationships are defined by the inner product between layers. A matrix of size $M \times N$ is constructed to represent this relationship, where each element (i, j) corresponds to the inner product between the $i - th$ channel of layer A and the $j - th$ channel of layer B. Yim et al. propose a two-stage method: first, adjusting the parameters of the small model based on the feature similarity preservation (FSP) matrix of the large model to align the layer relationships; then, continuing fine-tuning the small model parameters using the original loss function, such as cross-entropy. This approach aims to preserve the feature similarity between the two models while maintaining the original learning objective (Yim et al. 2017).

9.6 Inference Library for IoT Devices

The inference performance of on-device models is influenced by multiple factors, including hardware, models, and software, such as DL execution engines or libraries. DL libraries aim to enable on-device inference, and several major vendors have developed their own DL libraries, including TFLite (Haris et al. 2022), Core ML (Deng 2019), NCNN (Courville & Nia 2019), MNN (Jiang et al. 2020; Zhang et al. 2023), etc. TensorFlow and Caffe have been deprecated and replaced by their lightweight implementations, TFLite and PyTorchMobile, respectively. This work provides a summary of popular DL libraries such as TFLite (Haris et al. 2022), PyTorchMobile, NCNN (Courville & Nia 2019), MNN (Jiang et al. 2020), MACE (Lebedev & Belecky 2021), and SNPE (Zhang et al. 2022). Table 9.1 presents

Table 9.1 A comparison of representative DL libraries on mobile devices

Library	Developer	CPU FP32	CPU INT8	GPU FP32	GPU INT8	DSP INT8
TFlite	Google	✓	✓	✓	✓	✓
Pytorch Mobile	Facebook	✓	✓			
NCNN	Tencent	✓	✓	✓	✓	
MNN	Alibaba	✓	✓	✓	✓	
MACE	Xiaomi	✓	✓	✓	✓	
SNPE	Qualcomm	✓	✓	✓	✓	✓

a comparison of these DL libraries, considering their support for various model precision and hardware configurations.

Current DL libraries often do not fully leverage the capabilities of different hardware platforms. Each DL library typically supports at least one hardware platform, such as CPU, although PyTorchMobile lacks GPU acceleration support (Courville & Nia 2019). Interestingly, the DL library that achieves the best performance for a given model can vary depending on the specific hardware used. This difference in inference performance can be attributed to two main factors. Firstly, the hardware ecosystem exhibits a high level of fragmentation due to variations in architecture, such as Big. Little Core, cache size, GPU capacity, etc. Secondly, the heterogeneity of model structures also plays a role. The implementation of depth-wise convolution operators, for example, differs significantly from that of traditional convolution operators, as they have distinct cache access patterns (Zhang et al. 2022).

Even when using the same GPU, DL libraries provide different backend options. For example, MNN offers three backends: Vulkan, OpenGL, and OpenCL (Jiang et al. 2020). Interestingly, different backend choices can be more suitable for different models and devices. This may seem unexpected since MNN's Vulkan backend is primarily designed for cross-platform compatibility, including desktop, while OpenGL and OpenCL are mobile-specific programming interfaces that are highly optimized for mobile devices. This phenomenon can be attributed to both the underlying design of these backends and how DL developers implement the DL operators on top of them.

TFLite and SNPE provide acceleration capabilities for INT8 models running on DSP. For example, Qualcomm DSP is equipped with AI capabilities, such as HTA and HTP (Zhang et al. 2022), which are integrated with Hexagon vector extension (HVX) acceleration. The Winograd algorithm is also utilized to accelerate convolution calculations on the DSP. Furthermore, the energy-saving benefits of the DSP are particularly significant compared to the speed of inference.

To achieve optimal performance when executing models on devices, developers often need to integrate multiple DL libraries and dynamically select the appropriate one based on the current model and hardware platform. However, this approach is seldom implemented in practice due to the substantial overhead in terms of software complexity and development efforts. There is a need for a more lightweight system that can efficiently leverage the best performance from different DL libraries.

9.7 Inference Systems for IoT Devices

A considerable amount of research has been dedicated to exploring the collaborative utilization of local and cloud resources for inference. In contrast to the traditional completely offloading tasks to the server, these studies leverage the inherent characteristics of CNNs to optimize the offloading process (Donta & Dustdar 2022). Existing literature addresses the offloading of CNN inference to various destinations, including IoT devices within the local network (Xu et al. 2020; Mao et al. 2017), third-party IoT devices that possess the CNN computational graph (Almeida et al. 2022), and a choice between devices and servers through model selection (Han et al. 2016). Although these research endeavors share close relationships, the systems developed in these studies often have distinct requirements, such as the inclusion of multiple devices in a local area network, diverse optimization goals, such as distributing computations in a share-nothing setup, or significant overhead associated with maintaining multiple models.

9.7.1 Edge Cache-Based Inference

DL applications typically depend on real-time data provided by IoT devices, demonstrating specific similarity traits: (1) Temporal similarity: sequential frames in video streams captured by cameras frequently reveal similarities, such as consistent background or scene elements. (2) Spatial similarity: individuals' daily movement patterns, such as recurring travel between places like a laboratory and a restaurant, often show a high degree of repetition. Although variations in the captured images can occur due to alterations in lighting or background, image feature extraction algorithms like Speeded Up Robust Features (SURF) can capture these disparities (Xu et al. 2021). Note that despite the pronounced similarity between frames, they are not identical. The recurrent use of identical data can lead to a decrease in model accuracy. Consequently, the effective use of caching strategies is crucial to maintaining a balance between accuracy and efficiency. In this regard, there are some representative properties as follows.

Starfish enables the execution of multiple mobile vision applications on wearable devices while effectively managing computing resources and shared memory (LiKamWa & Zhong 2015). Its workflow is shown in Fig. 9.5. Researchers initially identified the need to parallelize multiple vision applications on existing wearable devices for simultaneous recognition and prediction tasks. However, these algorithms often rely on common data preprocessing steps such as grayscale processing, size adjustment, and feature extraction. Executing these algorithms separately on the same input image by different applications leads to redundant operations and memory consumption. To address this, Starfish decouples the CV library from the application, running it as an independent process (Core). API calls are transformed into cross-process calls, allowing the Core process to handle CV library calls

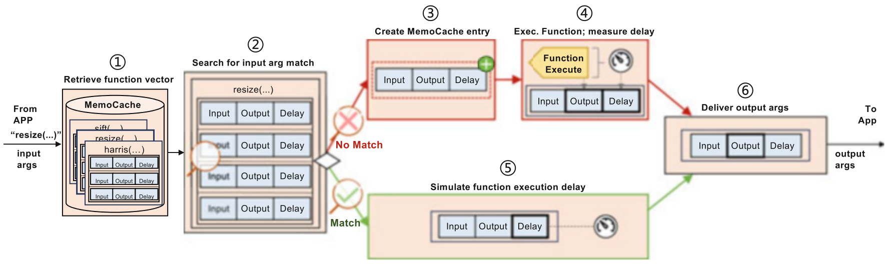

Fig. 9.5 The overview of Starfish framework

from all vision applications. A cache mechanism is employed to reduce redundant calculations and storage, with the API called by the CV library being key cached elements. Starfish, built on Android and optimized for the OpenCV (Bradski et al. 2000) library, effectively mitigates redundant operations and enhances resource utilization.

DeepMon leverages caching in mobile CV to enhance the performance of CNN inference (Huynh et al. 2017). The camera captures a series of video streams, and each frame is processed to convert it into an information stream that is then outputted to the users. The authors observe that consecutive frames in the video stream captured by IoT devices, such as smartphones and smart glasses, often contain significant pixel similarities, as vision applications typically require the camera to remain focused on a specific scene (e.g., object recognition, selfies) for a certain duration. Exploiting this pixel similarity, the author proposes using it as a cache to reduce the computation required by the CNN. However, traditional CNNs are considered a black box, where the output is obtained directly from the input image. The nature of convolution operations necessitates the dependence of output results on each frame's input image. To address this challenge, the authors redesign the forward algorithm of each CNN layer, enabling the (partial) computation results from the previous frame to be reused in the intermediate calculation process.

Guo and Hu (2018) incorporate a cross-application caching mechanism to minimize redundant calculations on the same or similar sensor input data, such as images. For instance, when both an image recognition application and an AR application utilize the same CNN model for image recognition, the output of CNN models can be cached. The cache lookup key is a variable-length feature vector that developers can specify, such as SIFT, SURF, etc. Furthermore, Potluck assigns an importance value to each cache item, indicating its frequency of use and potential time-saving benefits. Given the limited storage capacity, priority is given to caching the more important items. To find the most suitable cache entry, Potluck employs a nearest neighbor algorithm. The authors implement Potluck as a background service on the Android system and demonstrate its ability to reduce computing delay by up to 2.5–10× when used in applications.

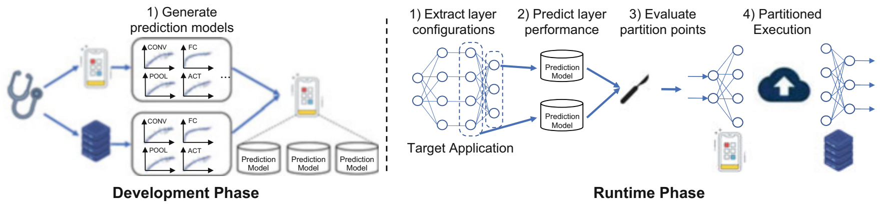
1) Generate prediction models
CONV
FC
POOL
ACT
Prediction Model
Development Phase
1) Extract layer configurations
2) Predict layer performance
3) Evaluate partition points
4) Partitioned Execution
Target Application
Runtime Phase

Fig. 9.6 The Neurosurgeon (Kang et al. 2017) framework tailored for computing offloading-based framework

9.7.2 Computing Offloading-Based Inference

Table 9.2 presents a comprehensive comparison of inference systems based on computing offloading. Note that these systems are specifically designed for a particular class of CNNs that are optimized for tasks such as classification and object detection. Among these systems, one notable work in this area is Neurosurgeon (Kang et al. 2017), as illustrated in Fig. 9.6, a framework that focuses on selecting an optimal split point to offload models between the devices and servers, with the aim of minimizing latency or energy consumption. However, the evaluation of Neurosurgeon primarily involves simple sequential CNN models, and the offloading decisions tend to be polarized, either offloading nothing or offloading everything, depending on the network conditions. Another relevant work is Hu et al. (2019), which introduces a scheduling scheme for partitioning DNNs under different network conditions to minimize either overall latency or throughput. However, it should be noted that the proposed scheduler lacks support for SLO deadlines, which are important in real-time applications. MCDNN (Han et al. 2016) is a framework that enables parallel computing for multiple applications. It uses a shared feature extraction layer and dynamically selects smaller, faster models to optimize accuracy and efficiency. The model selection is based on model catalogs, allowing for flexible adaptation to task requirements and available resources.

In terms of the compression of transferred data, JALAD (Li et al. 2018), SPINN (Laskaridis et al. 2020), and DynO (Almeida et al. 2022) incorporate the quantization scheme. However, it should be noted that SPINN utilizes a fixed 8-bit quantization level that is uniform across the split layers, without considering the dynamic range of the data or the resilience of each layer to quantization. DynO's compression method includes the compression method used in SPINN. DynO offers greater adaptability by dynamically selecting the optimal combination of bitwidth and split points based on performance targets and networking conditions. On the other hand, JALAD utilizes a decoupled DL model to make offloading decisions using a joint accuracy- and latency-aware execution framework (Li et al. 2018). However, JALAD is associated with a significant accuracy drop to achieve performance improvements. Additionally, JALAD only provides static configurations and lacks the ability to adapt to dynamic network conditions during runtime, limiting its efficiency on resource-constrained devices.

Table 9.2 Comparison of the existing inference systems

Work	Model	Network	Offloading granularity	Communication optimization	Scheduler	Decision variable(s)	Objectives
Neurosurgeon (Kang et al. 2017)	AlexNet, VGG, Deepcace	WiFi, LTE, 3G	Layer	×	Dynamic, SO, Exhaustive	Split point	Latency, Energy
JALAD (Li et al. 2018)	VGG, ResNet	LAN	Layer	Dynamic bitwidth	Static, SO, ILP	Split point, bitwidth	Latency
DADS (Hu et al. 2019)	YOLO, Inception	3G, 4G, WiFi	Layer	×	Dynamic, SO, Heuristic	Split point	Latency, Throughput
MoDNN (Mao et al. 2017)	VGG16	WLAN	neurons	×	Static, SO, Heuristic	Neuron partition	Latency
DeepThings (Zhao et al. 2018)	YOLOv2	WLAN	Cross-layer tile	×	Static, SO, Manual	Tile size, no. of layers	Latency, Throughput
MCDNN (Han et al. 2016)	VGG16	WLAN	Model	×	Dynamic, MO, Heuristic	Model variant, cloud or device	Latency, Energy
Clio (Huang et al. 2020)	MobileNetV2, VGG, ResNet	LORA, ZigBee, BLE, WiFi	Layer	Dynamic width	Dynamic, SO, Exhaustive	Split point, cloud-model width	Latency
ELF (Xu et al. 2020)	FastRCNN	WiFi	Image patch	×	Dynamic, SO, Multi-server	Patch packing, server allocation	Latency
IONN (Jeong et al. 2018)	Alexnet, Inception, ResNet	WLAN	Layer	×	Dynamic, SO, Heuristic	Split point	Latency
Edgnt (Li et al. 2018)	ALexNet	4G,LTE	Layer	EE	Dynamic, SO, Exhaustive	Split point, model depth	Latency
SPINN (Laskaridis et al. 2020)	Resnet50, Resnet56, mobileNetV2	4G,LTE	Layer	Fixed(8-bit) + EE	Dynamic, MO, Exhaustive	Split point, EE-polict	Latency, Throughput, Accuracy
DynO (Almeida et al. 2022)	mobileNetV2, ResNet152, InceptionV3	4G,WiFi	Layer	ISQuant + BitShuffling + LZ4	Dynamic, MO, Exhaustive	Split point, bitwidth	Server cost, Latency, Throughput, Accuracy

Clio (Huang et al. 2020) and SPINN (Laskaridis et al. 2020) focus on different aspects of model offloading: Clio considers the width of models, while SPINN focuses on the depth. However, these approaches require additional training for early classifiers or slicing-aware schemes, leading to increased computational overhead for pre-trained models. In contrast, DynO can directly target any pre-trained models without incurring additional costs. DynO proposes a distributed CNN inference framework that splits the computation between the client device and a more powerful remote end. It utilizes an online scheduler to optimize latency and throughput and minimize cloud workload and associated costs through device-offloading policies.

Early-Exit (EE)-based inference is a strategy that allows for accelerated inference by implementing early exits from specific branches within a model, capitalizing on the observation that early layers of models often capture significant features. One example of this approach is BranchyNet, which introduces supplementary side branches in addition to the main branch of the model (Teerapittayanon et al. 2016). BranchyNet enables the early termination of the inference process at an earlier layer when certain conditions are satisfied, resulting in substantial computational savings. It dynamically selects the branch that achieves the shortest inference time while maintaining a specified level of accuracy. By incorporating additional side branch classifiers, EE-based inference allows for early termination when processing easier samples with high confidence, while more challenging samples utilize more or all layers to ensure accurate predictions. This adaptive approach optimizes both inference speed and accuracy based on the characteristics of the input data. Another work, Edgent (Li et al. 2018), integrates BranchyNet to resize DNNs and enhance the efficiency of the inference process. By reducing the latency requirement, Edgent dynamically adjusts the optimal exit point in BranchyNet, resulting in improved accuracy. Additionally, Edgent utilizes adaptive partitioning, enabling collaborative and on-demand co-inference of DNNs.

In addition, another approach in this field focuses on exploiting the variability in the difficulty of different inputs to adapt the computations. Various works have been done in this area, including dynamic DNNs that adjust the depth of models (Panda et al. 2016; Kouris et al. 2022; Laskaridis et al. 2020; Panda et al. 2016), dynamic channel pruning (Jayakodi et al. 2020), or progressive inference schemes for generative adversarial network-based image generation (Jayakodi et al. 2020). These approaches offer flexibility in tuning the trade-off between accuracy and efficiency in the inference system.

9.8 Challenges and Opportunities of Inference

Despite the aforementioned benefits, the implementation of inference for IoTs still encounters various challenges and presents opportunities, as outlined below:

Model Optimization. With the advent of large models, how to run large sizes such as transformer-based models on IoT devices is an interesting direction. The acceleration of quantized models in inference is not universal and depends on the involved hardware (Zhang et al. 2022). However, there is significant potential to improve the inference speed of quantized models through software optimizations. By automating the compression, researchers can explore algorithms and strategies to effectively balance the trade-off between model size reduction and inference performance. On the other hand, the development trend for lightweight models on IoT devices is to achieve a similar trade-off in inference.

Algorithm-Hardware Codesign. The lightweight DL models and compression techniques should take into consideration the underlying hardware architecture, enabling hardware-algorithm codesign to achieve more efficient inference. DL developers should prioritize optimization for heterogeneous processors (Guo et al. 2023), expanding support for various types of operators and enhancing single-operation performance. In many scenarios, more powerful CPUs and accelerators, especially GPUs and DSPs, can significantly accelerate inference (Zhang et al. 2023). This encourages DL researchers to design models well-suited for GPU computing, emphasizing operators with high parallelism while minimizing memory-intensive operations that hinder parallelism.

Neural Network Hardware Accelerator. To design a reasonable scheduling mode in a complex and multi-application operating environment, it is vital to consider the relatively primitive driver management compared to CPU and GPU. There is a significant research opportunity to address this gap by introducing flexibility in SoCs to effectively handle and adapt to the evolving requirements of improved DL operations. The addition of flexibility can enhance silicon efficiency and lead to cost-friendly solutions. Consequently, the integration of dynamic reconfigurability into SoCs is expected. However, it is crucial to minimize power consumption and area in SoCs that incorporate extra logic. Therefore, research efforts focused on reducing power consumption and optimizing the area of such SoCs are actively pursued.

DL Library Selection. It is crucial to assess the advantages and disadvantages of various DL libraries and devise a solution that can unify their strengths. Otherwise, the issue of inference performance fragmentation may persist for an extended period, as resolving it requires substantial engineering efforts. Achieving optimal performance in mobile DL applications often necessitates the integration of multiple DL libraries and dynamic loading based on the current model and hardware platform. However, this approach is seldom implemented due to the considerable overhead in terms of software complexity and development efforts. There is a need for a more lightweight system that can harness the superior performance of different DL libraries.

Developing Benchmarks. Proper benchmark standards are crucial for accurately evaluating the inference performance (Ren et al. 2023). To enable meaningful comparisons of DL models, optimization algorithms, and hardware platforms, a universal and comprehensive set of quality metrics specific to inference is essential. Currently, benchmark datasets and models predominantly focus on CNNs evaluated on the ImageNet (Azizi et al. 2023). To ensure a more comprehensive evaluation, it is necessary to develop additional benchmark datasets, libraries, and DL models that cover a wider range of applications and input data types. This will facilitate a more thorough assessment of inference performance.

Explainability in Inference. The adoption of AI in critical domains has raised concerns about transparency and accountability. Explainable AI (XAI) aims to address these concerns by providing transparency in DL algorithms (Adadi & Berrada 2018). However, ensuring explainability in the context of intelligent inference remains a challenging and underexplored research area, particularly considering the trade-off between optimization and accuracy. Resolving this challenge is essential for the responsible deployment of AI.

Complex Audio Processing Models. Most existing research focuses predominantly on image-processing tasks, leaving a notable gap in the exploration of similar methods for audio-processing models. However, we posit that the techniques developed for image processing can also be effectively applied to these audio scenarios. Specifically, when addressing RNN-based models, such as LSTMs (Yu et al. 2019) and GRUs (Jiao et al. 2020), their recurrent nature introduces dependencies between samples that are absent in CNNs. Consequently, this poses a challenge in offloading computations, as the RNNs must be transferred alongside the computation. While the partitioning strategies employed in prior studies demonstrate applicability to various DNN architectures by automatically identifying split-point dependencies, RNNs necessitate specialized treatment. The future of inference systems is expected to encompass a wide range of architectures and use cases, showcasing their versatility and applicability in various domains.

Resource Allocation for Inference. The collaborative DNN inference application scenarios are characterized by dynamic environments where future events are challenging to predict accurately. To effectively handle large-scale tasks, it is crucial to have robust online edge resource coordination and provisioning capabilities (Donta et al. 2023; Adadi & Berrada 2018; Dustdar & Murturi 2020; Alkhabbas et al. 2020; Tsigkanos et al. 2019). Real-time joint optimization of heterogeneous computing, communication, and cache resource allocation, along with customized system parameter configuration based on task requirements, is necessary. Addressing the complexity of algorithm design, an emerging research direction focuses on efficient resource allocation strategies driven by data-driven adaptive learning.

Enhancing Security in Inference. Ensuring the credibility of services in distributed collaborative inference requires the design of a robust distributed security mechanism (Flamis et al. 2021; Sedlak et al. 2022). This mechanism plays a vital role in authenticating subscribers, controlling access to collaborative inference tasks, ensuring model and data security on devices, and facilitating mutual authentication between different devices. Furthermore, ongoing research explores the use of blockchain technology to enhance the security and privacy of devices and data in collaborative inference. This avenue holds promise and warrants further exploration in future collaborative DNN inference, particularly regarding privacy issues.

9.9 Conclusion

This chapter provides a comprehensive review of the current state of DL operating on IoT devices. It discusses various methods for accelerating DL inference across devices, edge servers, and the cloud, highlighting their utilization of the unique structure of DNN models and the geospatial locality of user requests in edge computing. The analysis emphasizes the crucial trade-offs between accuracy, latency, and energy that need to be considered. Despite significant progress, numerous challenges persist, including performance improvements, hardware and software optimization, resource management, benchmarking, and integration with other networking technologies. These challenges can be overcome through technological innovations in algorithms, system design, and hardware accelerations. As DL innovation continues at a rapid pace, it is anticipated that new technical challenges in edge computing will arise, providing further opportunities for innovation. Ultimately, this review aims to stimulate discussion, attract attention to the field of inference, and inspire future research endeavors.

References

Adadi, Amina, and Mohammed Berrada. 2018. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE access* 6: 52138–52160.

Ain, Qurat-ul et al. 2018. IoT operating system based fuzzy inference system for home energy management system in smart buildings. *Sensors* 18 (9): 2802.

Alkhabbas, Fahed, et al. 2020. A goal-driven approach for deploying self-adaptive IoT systems. In *2020 IEEE International Conference on Software Architecture (ICSA)*, 146–156. Piscataway: IEEE.

Almeida, Mario, et al. 2022. Dyno: Dynamic onloading of deep neural networks from cloud to device. *ACM Transactions on Embedded Computing Systems* 21 (6): 1–24.

Azizi, Shekoofeh, et al. 2023. Synthetic data from diffusion models improves imagenet classification. arXiv preprint. arXiv:2304.08466.

Bajrami, Xhevahir, et al. 2018. Face recognition performance using linear discriminant analysis and deep neural networks. *International Journal of Applied Pattern Recognition* 5 (3): 240–250.

Bradski, Gary, Adrian Kaehler, et al. 2000. OpenCV. *Dr. Dobb's Journal of Software Tools* 3 (2): 1–81.

Cheng, Yu, et al. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, 2857–2865.

Choudhary, Tejalal, et al. 2020. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review* 53: 5113–5155.

Courville, Vanessa, and Vahid Partovi Nia. 2019. Deep learning inference frameworks for ARM CPU. *Journal of Computational Vision and Imaging Systems* 5 (1): 3–3.

Deng, Yunbin. 2019. Deep learning on mobile devices: A review. In *Mobile Multimedia/Image Processing, Security, and Applications 2019*. Vol. 10993, 52–66. Bellingham: SPIE.

Developers, TensorFlow. 2022. TensorFlow. In *Zenodo*.

Donta, Praveen Kumar, and Schahram Dustdar. 2022. The promising role of representation learning for distributed computing continuum systems. In *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 126–132. Piscataway: IEEE.

Donta, Praveen Kumar, Boris Sedlak, et al. 2023. Governance and sustainability of distributed continuum systems: A big data approach. *Journal of Big Data* 10 (1): 1–31.

Dustdar, Schahram, and Ilir Murturi. 2020. Towards distributed edge-based systems. In *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, 1–9. Piscataway: IEEE.

Dustdar, Schahram, and Ilir Murturi. 2021. Towards IoT processes on the edge. In *Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future: Essays Dedicated to Michael Papazoglou on the Occasion of His 65th Birthday and His Retirement*, 167–178.

Flamis, Georgios, et al. 2021. Best practices for the deployment of edge inference: The conclusions to start designing. *Electronics* 10 (16): 1912.

Girshick, Ross, et al. 2015. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38 (1): 142–158.

Guo, Anqi, et al. 2023. Software-hardware co-design of heterogeneous SmartNIC system for recommendation models inference and training. In *Proceedings of the 37th International Conference on Supercomputing*, 336–347.

Guo, Peizhen, and Wenjun Hu. 2018. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 271–284.

Han, Seungyeop, et al. 2016. MCDNN: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 123–136.

Haris, Jude, Gibson, Perry, Cano, José, Agostini, Nicolas Bohm and Kaeli, David. 2022. Hardware/Software Co-Design of Edge DNN Accelerators with TFLite. 107 (8): 1–4.

Hu, Chuang, et al. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, 1423–1431. Piscataway: IEEE.

Huang, Jin, et al. 2020. Clio: Enabling automatic compilation of deep learning pipelines across iot and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 1–12.

Huynh, Loc N., et al. 2017. DeepMon: Mobile GPU-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 82–95.

Iandola, Forrest N., et al. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and$<$ 0.5 MB model size. arXiv preprint. arXiv:1602.07360.

Jayakodi, Nitthilan Kanappan, Janardhan Rao Doppa, et al. 2020. SETGAN: Scale and energy trade-off gans for image applications on mobile platforms. In *Proceedings of the 39th International Conference on Computer-Aided Design*, 1–9.

Jayakodi, Nitthilan Kanappan, Syrine Belakaria, et al. 2020. Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models. *ACM Transactions on Embedded Computing Systems (TECS)* 19 (1): 1–24.

Jeong, Hyuk-Jin, et al. 2018. IONN: Incremental offloading of neural network computations from mobile devices to edge servers. In *Proceedings of the ACM Symposium on Cloud Computing*, 401–411.

Jiang, Xiaotang, et al. 2020. MNN: A universal and efficient inference engine. In *Proceedings of Machine Learning and Systems*. Vol. 2, 1–13.

Jiao, Meng, et al. 2020. A GRU-RNN based momentum optimized algorithm for SOC estimation. *Journal of Power Sources* 459: 228051.

Kang, Yiping, et al. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45 (1): 615–629.

Kounoudes, Alexia Dini et al. 2021. User-centred privacy inference detection for smart home devices. *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, 210–218. Piscataway: IEEE.

Kouris, Alexandros, et al. 2022. Multi-exit semantic segmentation networks. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXI*, 330–349. Berlin: Springer.

Laskaridis, Stefanos, Stylianos I. Venieris, Hyeji Kim, et al. 2020. HAPI: Hardware-aware progressive inference. In *Proceedings of the 39th International Conference on Computer-Aided Design*, 1–9.

Laskaridis, Stefanos, Stylianos I. Venieris, Mario Almeida, 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*, 1–15.

Lebedev, Mikhail, and Pavel Belecky. 2021. A survey of open-source tools for FPGA-based inference of artificial neural networks. In *2021 Ivannikov Memorial Workshop (IVMEM)*, 50–56. Piscataway: IEEE.

Lebedev, Vadim, and Victor Lempitsky. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2554–2564.

Leiserson, Charles E., et al. 2020. There's plenty of room at the Top: What will drive computer performance after Moore's law? *Science* 368 (6495): eaam9744.

Leon, Vasileios, et al. 2022. Systematic embedded development and implementation techniques on intel myriad VPUs. In *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 1–2. Piscataway: IEEE.

Li, En, et al. 2018. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*, 31–36.

Li, Hongshan, et al. 2018. 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)*, 671–678. Piscataway: IEEE.

Li, Liangzhi, et al. 2018. Deep learning for smart industry: Efficient manufacture inspection system with fog computing. *IEEE Transactions on Industrial Informatics* 14 (10): 4665–4673.

Li, Ying, et al. 2023. Federated domain generalization: A survey. arXiv preprint. arXiv:2306.01334.

LiKamWa, Robert, and Lin Zhong. 2015. Starfish: Efficient concurrency support for computer vision applications. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 213–226.

Liu, Hongye, et al. 2016. Deep relative distance learning: Tell the difference between similar vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2167–2175.

Liu, Shaoshan, et al. 2019. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE* 107 (8): 1697–1716.

Lucas, Bruce D., and Takeo Kanade. 1981. An iterative image registration technique with an application to stereo vision. In *IJCAI'81: 7th International Joint Conference on Artificial Intelligence*. Vol. 2, 674–679.

Mao, Jiachen, et al. 2017. MoDNN: Local distributed mobile computing system for deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 1396–1401. Piscataway: IEEE.

Mohammadi, Mehdi, and Ala Al-Fuqaha. 2018. Enabling cognitive smart cities using big data and machine learning: Approaches and challenges. *IEEE Communications Magazine* 56 (2): 94–101.

Owens, John D., et al. 2008. GPU computing. In *Proceedings of the IEEE* 96 (5): 879–899.

Panda, Priyadarshini, et al. 2016. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 475–480. Piscataway: IEEE.

Polino, Antonio, et al. 2018. Model compression via distillation and quantization. arXiv preprint. arXiv:1802.05668.

Rastegari, Mohammad, et al. 2016. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, 525–542. Berlin: Springer.

Ren, Wei-Qing, et al. 2023. A survey on collaborative DNN inference for edge intelligence. In *Machine Intelligence Research*, 1–25.

Romero, Adriana, et al. 2014. Fitnets: Hints for thin deep nets. arXiv preprint. arXiv:1412.6550.

Sedlak, Boris, et al. 2022. Specification and operation of privacy models for data streams on the edge. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, 78–82. Piscataway: IEEE.

Sengupta, Abhronil, et al. 2019. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience* 13: 95.

Soto, José Angel Carvajal, et al. 2016. CEML: Mixing and moving complex event processing and machine learning to the edge of the network for IoT applications. In *Proceedings of the 6th International Conference on the Internet of Things*, 103–110.

Sun, Yi, Chen, Yuheng, Wang, Xiaogang, Tang, Xiaoou. 2014. Deep learning face representation by joint identification-verification. *Advances in Neural Information Processing Systems* 27 (8): 1–8.

Targ, Sasha, et al. 2016. Resnet in resnet: Generalizing residual architectures. arXiv preprint. arXiv:1603.08029.

Teerapittayanon, Surat, et al. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2464–2469. Piscataway: IEEE.

Tsigkanos, Christos, et al. 2019. Dependable resource coordination on the edge at runtime. *Proceedings of the IEEE* 107 (8): 1520–1536.

Viola, Paul, and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1, I–I. Piscataway: IEEE.

Wang, Qipeng, et al. 2022. Melon: Breaking the memory wall for resource-efficient on-device machine learning. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 450–463.

Wang, Yang, et al. 2017. Effective multi-query expansions: Collaborative deep networks for robust landmark retrieval. *IEEE Transactions on Image Processing* 26 (3): 1393–1404.

Wu, Jiaxiang, et al. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4820–4828.

Xiang, Yecheng, and Hyoseung Kim. 2019. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, 392–405. Piscataway: IEEE.

Xu, Daliang, et al. 2022. Mandheling: Mixed-precision on-device DNN training with DSP offloading. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 214–227.

Xu, Mengwei, Jiawei Liu, et al. 2019. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*, 2125–2136.

Xu, Mengwei, Tiantu Xu, et al. 2021. Video analytics with zero-streaming cameras. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 459–472.

Xu, Mengwei, Xiwen Zhang, et al. 2020. Approximate query service on autonomous iot cameras. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 191–205.

Yim, Junho, 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*, 4133–4141.

Yu, Yong, et al. 2019. A review of recurrent neural networks: LSTM cells and network architectures. *Neural Computation* 31 (7): 1235–1270.

Zhang, Qiyang, Xiang Li, et al. 2022. A comprehensive benchmark of deep learning libraries on mobile devices. In *Proceedings of the ACM Web Conference 2022*, 3298–3307.

Zhang, Qiyang, Zuo Zhu, et al. 2023. Energy-efficient federated training on mobile device. *IEEE Network* 35 (5): 1–14.

Zhang, Xiangyu, et al. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6848–6856.

Zhao, Zhuoran, et al. 2018. Deepthings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37 (11): 2348–2359.

Zhou, Kanglei, et al. 2022. TSVMPath: Fast regularization parameter tuning algorithm for twin support vector machine. *Neural Processing Letters* 54 (6): 5457–5482.