1

Introduction

Deep neural networks (DNNs) have enabled the deployment of artificial intelligence (AI) in many modern applications including autonomous driving [1], image recognition [2], and speech processing [3]. In many applications, DNNs have achieved close to human-level accuracy and, in some, they have exceeded human accuracy [4]. This high accuracy comes from a DNN's unique ability to automatically extract high-level features from a huge quantity of training data using statistical learning and improvement over time. This learning over time provides a DNN with an effective representation of the input space. This is quite different from the earlier approaches where specific features were hand-crafted by domain experts and were subsequently used for feature extraction.

Convolutional neural networks (CNNs) are a type of DNNs, which are most commonly used for computer vision tasks. Among different types of DNNs, such as multilayer perceptrons (MLP), recurrent neural networks (RNNs), long short-term memory (LSTM) networks, radial basis function networks (RBFNs), generative adversarial networks (GANs), restricted Boltzmann machines (RBMs), deep belief networks (DBNs), and autoencoders, CNNs are the mostly commonly used. Invention of CNNs has revolutionized the field of computer vision and has enabled many applications of computer vision to go mainstream. CNNs have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, object detection, activity recognition, natural language processing, brain–computer interfaces, and financial time-series prediction.

DNN/CNN processing is usually carried out in two stages, training and inference, with both of them having their own computational needs. Training is the process where a DNN model is trained using a large application-specific data set. The training time is dependent on the model size and the target accuracy requirements. For high accuracy applications like autonomous driving, training a DNN can take weeks and is usually performed on a cloud. Inference, on the other

Accelerators for Convolutional Neural Networks, First Edition.
Arslan Munir, Joonho Kong, and Mahmood Azhar Qureshi.

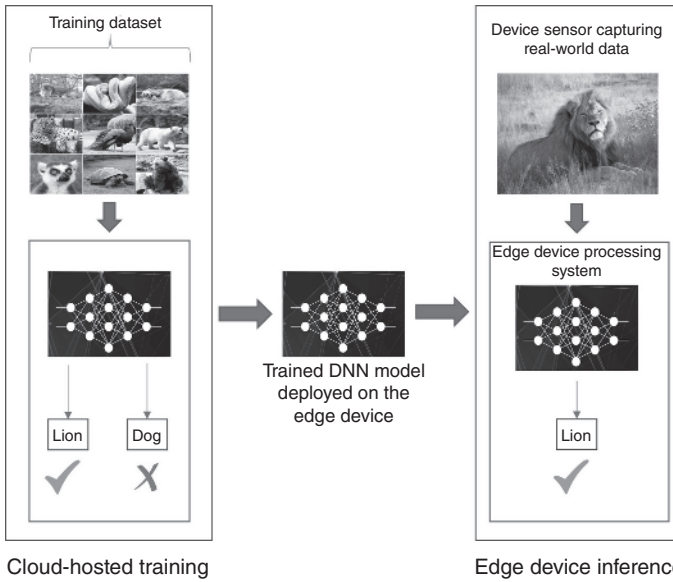

Figure 1.1 DNN/CNN processing methodology. Source: (b) Daughter#3 - Cecil/Wikimedia Commons/CC BY-SA 2.0.

hand, can be performed either on the cloud or the edge device (mobile device, Internet of things (IoT), autonomous vehicle, etc.). Nowadays, in many applications, it is advantageous to perform the inference process on the edge devices, as shown in Figure 1.1. For example, in cellphones, it is desirable to perform image and video processing on the device itself rather than sending the data over to the cloud for processing. This methodology reduces the communication cost and the latency involved with the data transmission and reception. It also eliminates the risk of losing important device features should there be a network disruption or loss of connectivity. Another motivation for doing inference on the device is the ever-increasing security risk involved with sending personalized data, including images and videos, over to the cloud servers for processing. Autonomous driving systems which require visual data need to deploy solutions to perform inference locally to avoid latency and security issues, both of which can result in a catastrophe, should an undesirable event occurs. Performing DNN/CNN inference on the edge presents its own set of challenges. This stems from the fact that the embedded platforms running on the edge devices have stringent cost limitations which limit their compute capabilities. Running compute and memory-intensive DNN/CNN inference in these devices in an efficient manner becomes a matter of prime importance.

1.1 History and Applications

Neural nets have been around since the 1940s; however, the first practically applicable neural network, referred to as the LeNet [5], was proposed in 1989. This neural network was designed to solve the problem of digit recognition in hand-written numeric digits. It paved the way for the development of neural networks responsible for various applications related to digit recognition, such as an automated teller machine (ATM), optical character recognition (OCR), automatic number plate recognition, and traffic signs recognition. The slow growth and a little to no adoption of neural networks in the early days is mainly due to the massive computational requirements involved with their processing which limited their study to theoretical concepts.

Over the past decade, there has been an exponential growth in the research on DNNs with many new high accuracy neural networks being deployed for various applications. This has only been possible because of two factors. The first factor is the advancements in the processing power of semiconductor devices and technological breakthroughs in computer architecture. Nowadays, computers have significantly higher computing capability. This enables the processing of a neural network within a reasonable time frame, something that was not achievable in the early days. The second factor is the availability of a large amount of training datasets. As neural networks learn over time, providing huge amounts of training data enables better accuracy. For example, Meta (parent company of Facebook) receives close to a billion user images per day, whereas YouTube has 300 hours of video uploaded every minute [6]. This enables the service providers to train their neural networks for targeted advertising campaigns bringing in billions of dollars of advertising revenue. Apart from their use in social media platforms, DNNs are impacting many other domains and are making a huge impact. Some of these areas include:

- **Speech Processing**: Speech processing algorithms have improved significantly in the past few years. Nowadays, many applications have been developed that use DNNs to perform real-time speech recognition with unprecedented levels of accuracy [3, 7–9]. Many technology companies are also using DNNs to perform language translation used in a wide variety of applications. Google, for example, uses Google's neural machine translation system (GNMT) [10] which uses LSTM-based seq2seq model for their language translation applications.
- **Autonomous Driving**: Autonomous driving has been one of the biggest technological breakthroughs in the auto industry since the invention of the internal combustion engine. It is not a coincidence that the self-driving boom came at the same time when high accuracy CNNs became increasingly popular. Companies

like Tesla [11] and Waymo [12] are using various types of self-driving technology including visual feeds and Lidar for their self-driving solutions. One thing which is common in all these solutions is the use of CNNs for visual perception of the road conditions which is the main back-end technology used in advanced driver assistance systems (ADAS).

- **Medical AI**: Another crucial area where DNNs/CNNs have become increasingly useful is medicine. Nowadays, doctors can use AI-assisted medical imagery to perform various surgeries. AI systems use DNNs in genomics to gather insights about genetic disorders like autism [13, 14]. DNNs/CNNs are also useful in the detection of various types of cancers like skin and brain cancer [15, 16].
- **Security**: The advent of AI has challenged many traditional security approaches that were previously deemed sufficient. The rollout of 5G technology has caused a massive surge of IoT-based deployments which traditional security approaches are not able to keep up with. Physical unclonability approaches [17–21] were introduced to protect this massive deployment of IoTs against security attacks with minimum cost overheads. These approaches, however, were also unsuccessful in preventing AI-assisted attacks using DNNs [22, 23]. Researchers have now been forced to upgrade the security threat models to incorporate AI-based attacks [24, 25]. Because of a massive increase in AI-assisted cyber-attacks on cloud and datacenters, companies have realized that the best way of defeating offensive AI attacks is by incorporating AI-based counterattacks [26, 27].

Overall, the use of DNNs, in particular CNNs, in various applications has seen exponential growth over the past decade, and this trend has been on the rise for the past many years. The massive increase in CNN deployments on the edge devices requires the development of efficient processing architectures to keep up with the computational requirements for successful CNN inference.

1.2 Pitfalls of High-Accuracy DNNs/CNNs

This section discusses some of the pitfalls of high-accuracy DNN/CNN models focusing on compute and energy bottlenecks, and the effect of sparsity of high-accuracy models on throughput and hardware utilization.

1.2.1 Compute and Energy Bottleneck

CNNs are composed of multiple convolution layers (CONV) which help in extracting low-, mid-, and high-level input features for better accuracy. Although CNNs are primarily used in applications related to image and video processing, they are

Table 1.1 Popular CNN models.

CNN model	Layers	Top-1 accuracy (%)	Top-5 accuracy (%)	Parameters	MACs
AlexNet [30]	8	63.3	84.6	62M	666M
VGG-16 [31]	16	74.3	91.9	138M	15.3B
GoogleNet [35]	22	68.9	88	6.8M	1.5B
MobileNet [35]	28	70.9	89.9	4.2M	569M
ResNet-50 [32]	50	75.3	92.2	25.5M	3.9B

also used in speech processing [3, 7], gameplay [28], and robotics [29] applications. We will further discuss the basics of CNNs in Chapter 2. In this section, we explore some of the bottlenecks when it comes to implementing *high-accuracy* CNN inference engines in embedded mobile devices.

The development of high accuracy CNN models [30–34] in recent years has strengthened the notion of employing DNNs in various AI applications. The classification accuracy of CNNs for the ImageNet challenge [2] has improved considerably from 63.3% in 2012 (AlexNet [30]) to a staggering 87.3% (EfficientNetV2 [4] in 2021). This high jump in accuracy comes with high compute and energy costs for CNN inference. Table 1.1 shows some of the most commonly used CNN models. The models are trained using the ImageNet dataset [2], and the top-1 and top-5 classification accuracy is also given. We note that top-1 accuracy is the conventional accuracy, which means that the model answer (i.e., the one predicted by the model with the highest probability) must be exactly the expected answer. Top-5 accuracy means that any of the five highest probability answers predicted by the model must match the expected answer. It can be seen from Table 1.1 that the addition of more layers results in better accuracy. This addition, however, also corresponds to a greater number of model parameters, requiring more memory and storage. It also results in higher multiply-accumulate (MAC) operations, causing an increase in computational complexity and resource requirements, which in turn, affects the performance of the edge devices.

Even though some efforts have been made to reduce the size of the high accuracy models, they still require massive amounts of computations over a series of network layers to perform a particular inference task (classification, segmentation, etc.). These tremendous number of computations (typically in tens of millions) present a huge challenge for the **neural network accelerators** (NNAs) running the CNN inference. NNAs are specialized hardware blocks inside a computer system (e.g., mobile devices and cloud servers) that speed up the computations of the CNN inference process to maintain the real-time requirements of the system

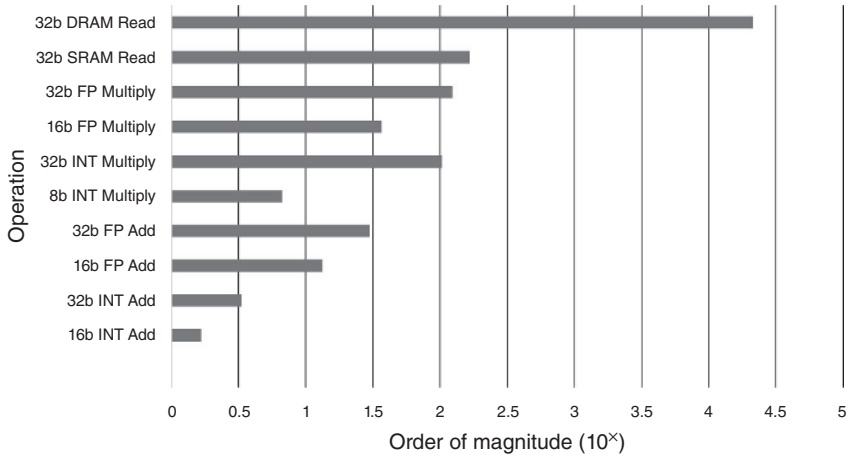

Figure 1.2 Energy cost (relative to 8 bit Add operation) shown on a log 10 scale for a 45 nm process technology. Source: Adapted from [6, 36].

and improve system throughput. Apart from the massive computational requirements, the addition of more layers for higher accuracy drastically increases the CNN model size. This prevents the CNN model from being stored in the limited on-chip static random access memory (SRAM) of the edge device, and, therefore, requires off-chip dynamic random access memory (DRAM) which presents a high DRAM access energy cost.

To put this in perspective, the energy cost per fetch for 32 bit coefficients in an off-chip low-power double data rate 2 (LPDDR2) DRAM is about 640 pJ, which is about 6400× the energy cost of a 32 bit integer ADD operation [36]. The bigger the model is, the more memory referencing is performed to access the model data which in turn expends more energy. Figure 1.2 shows the energy cost of various compute and memory operations relative to an 8 bit integer add (8 bit INT Add) operation. It can be seen that the DRAM Read operation dominates the energy graph with the 32 bit DRAM Read consuming greater than 4 orders of magnitude higher energy than the 8 bit INT Add. As a consequence, the energy cost from just the DRAM accesses would be well beyond the limitations of an embedded mobile device with limited battery life. Therefore, in addition to accelerating the compute operations, the NNA also needs to minimize the off-chip memory transactions for decreasing the overall energy consumption.

Many algorithm-level techniques have been developed to minimize the computational requirements of a CNN without incurring a loss in accuracy. Since the main compute bottleneck in CNN inference is the CONV operation, *Mobilenets* [33, 34] were developed to reduce the total number of CONV operations. These CNNs drastically reduce the total number of parameters and MAC operations by

breaking down the standard 2D convolution into depthwise separable and pointwise convolutions. The depthwise separable and pointwise convolutions result in 8× to 9× reduction in total computations compared to regular CONV operations, with a slight decrease in accuracy. They also eliminate varying filter sizes, and instead, use 3×3 and 1×1 filters for performing convolution operations. This makes them ideal for embedded mobile devices because of their relatively low memory footprint and lower total MAC operations.

A widely used approach for decreasing the memory bottleneck is the reduction in the precision of both weights and activations using various quantization strategies [37–39]. This again does not result in a significant loss in accuracy and reduces the model size by a considerable amount. Hardware implementations like Envision [40], UNPU [41], and Stripes [42] show how reduced bit precision, and quantization, translates into better savings in energy.

1.2.2 Sparsity Considerations

Nonlinear activation functions [6], in addition to deep layers, is one of the key characteristics that improve the accuracy of a CNN model. Typically, nonlinearity is added by incorporating activation functions, the most common being the rectified linear unit (ReLU) [6]. The ReLU converts all negative values in a feature map to zeros. Since the output of one layer is the input to the next layer, many of the computations, within a layer, involve multiplication with zeros. These feature maps containing zeros are referred to as *one-sided* sparse feature maps. The multiplications resulting from this one-sided sparsity waste compute cycles and decrease the *effective* throughput and hardware utilization, thus, reducing the performance of the accelerator. It also results in high energy costs as the transfer of zeros to/from off-chip memory is wasted memory access. In order to reduce the computational and memory access volume, previous works [43–45] have exploited this one-sided sparsity and displayed some performance improvements. To exacerbate the issue of wasted compute cycles and memory accesses, *two-sided* sparsity is introduced in CNNs often by pruning techniques when, in addition to the feature maps, the weight data also consists of zeros. Designing a CNN accelerator that can overcome the wasted compute cycles and memory accesses issues of one-sided and two-sided sparsities is quite challenging.

In recent years, many pruning techniques have been developed for the compression of DNN models [46–49]. Han et al. [46] iteratively pruned the connections based on parameter threshold and performed retraining to retain accuracy. This type of pruning is referred to as unstructured pruning. It arbitrarily removes weight connections in a DNN/CNN but does little to improve acceleration on temporal architectures like central processing units (CPUs) and graphics processing units (GPUs) which rely on accelerating matrix multiplications. Another form

of pruning, referred to as structured pruning [50, 51], reduces the size of weight matrices and maintains a full matrix. This makes it possible to simplify the NNA design since the sparsity patterns are predictable, therefore, enabling better hardware support for operation scheduling.

Both unstructured and structured pruning strategies, as described above, result in *two-sided* sparsity, (i.e., sparsity in both weights and activations) which lead to approximately 9× model reduction for AlexNet and 13× reduction for VGG-16. The purning strategies also result in 4–9× *effective* compute reduction (depending on the model). These gains seem very promising; however, designing an accelerator architecture to leverage them is quite challenging because of the following reasons:

- **Data Access Inconsistency**: Computation gating is one of the most common ways by which sparsity is generally exploited. Whenever a zero in the activation or the weight data is read, no operation is performed. This results in energy savings but has no impact on the throughput because of the wastage of compute cycle. Complex read logic needs to be implemented to discard the zeros, and instead, perform effective computations on nonzero data. Some previous works [52, 53] use sparse compression formats like compressed sparse column (CSC) or compressed sparse row (CSR) to represent sparse data. These formats have variable lengths and make *looking ahead* difficult if both the weight and the activation sparsity are being considered. Other than that, developing the complex control and read logic to process these formats can be quite challenging.
- **Low Utilization of the Processing Element (PE) Array**: Convolution operations for CNN inference are usually performed using an array of two-dimensional PEs in a CNN accelerator. Different dataflows (input stationary, output stationary, weight stationary, etc.) have been proposed that efficiently map the weight data and the activation data onto the PE array to maximize the throughput [6, 54]. Sparsity introduces inconsistency in the scheduling of data thereby reducing hardware utilization. The subset of PEs provided with more sparse data have idle times while those provided with less sparse (or denser) data are fully active. This bounds the throughput of the accelerator to the most active PEs, and therefore, leads to the underutilization of the PE array.

Considering the abovementioned issues, many accelerators have been proposed in the past that attempt to strike a balance between hardware resource complexity and performance improvements. The CNN accelerators that exploit sparsity in CNN models are covered in detail in Part IV of this book.

1.3 Chapter Summary

This chapter discussed the history and applications of DNNs, focusing on CNNs. The chapter also highlighted the compute and energy bottlenecks as well as the effect of sparsity in high-accuracy CNN models on the throughput and hardware utilization of edge devices.