Deep Neural Network Approximation for Custom Hardware: Where We've Been, Where We're Going

ERWEI WANG, JAMES J. DAVIS, RUIZHE ZHAO, and HO-CHEUNG NG,
Imperial College London
XINYU NIU, Corerain Technologies
WAYNE LUK, PETER Y. K. CHEUNG, and GEORGE A. CONSTANTINIDES,
Imperial College London

Deep neural networks have proven to be particularly effective in visual and audio recognition tasks. Existing models tend to be computationally expensive and memory intensive, however, and so methods for hardware-oriented approximation have become a hot topic. Research has shown that custom hardware-based neural network accelerators can surpass their general-purpose processor equivalents in terms of both throughput and energy efficiency. Application-tailored accelerators, when co-designed with approximation-based network training methods, transform large, dense, and computationally expensive networks into small, sparse, and hardware-efficient alternatives, increasing the feasibility of network deployment. In this article, we provide a comprehensive evaluation of approximation methods for high-performance network inference along with in-depth discussion of their effectiveness for custom hardware implementation. We also include proposals for future research based on a thorough analysis of current trends. This article represents the first survey providing detailed comparisons of custom hardware accelerators featuring approximation for both convolutional and recurrent neural networks, through which we hope to inspire exciting new developments in the field.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies** → **Neural networks**; • **Hardware** → **Hardware accelerators**;

Additional Key Words and Phrases: FPGAs, ASICs, approximation methods, convolutional neural networks, recurrent neural networks

ACM Reference format:
Erwei Wang, James J. Davis, Ruizhe Zhao, Ho-Cheung Ng, Xinyu Niu, Wayne Luk, Peter Y. K. Cheung, and George A. Constantinides. 2019. Deep Neural Network Approximation for Custom Hardware: Where We've Been, Where We're Going. *ACM Comput. Surv.* 52, 2, Article 40 (May 2019), 39 pages.
https://doi.org/10.1145/3309551

The support of the United Kingdom EPSRC (Grants No. EP/K034448/1, No. EP/P010040/1, No. EP/N031768/1, No. EP/I012036/1, No. EP/L00058X/1, and No. EP/L016796/1), European Union Horizon 2020 Research and Innovation Programme (Grant No. 671653), Corerain, Imagination Technologies, Intel, Maxeler, Royal Academy of Engineering, SGIIT, China Scholarship Council, and Lee Family Scholarship are gratefully acknowledged.
Authors' addresses: E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, Department of Electrical and Electronic Engineering, Imperial College London, London, SW7 2AZ, United Kingdom; emails: {erwei.wang13, james.davis, p.cheung, g.constantinides}@imperial.ac.uk; X. Niu, Corerain Technologies, Shenzhen, China; email: xinyu.niu@corerain.com; R. Zhao, H.-C. Ng, and W. Luk, Department of Computing, Imperial College London, London, United Kingdom; emails: {ruizhe.zhao15, h.ng16, w.luk}@imperial.ac.uk.

0360-0300/2019/05-ART40 $15.00
https://doi.org/10.1145/3309551

1 INTRODUCTION

The exponentially growing availability of digital data such as images, videos, and speech from myriad sources, including social media and the Internet of Things, is driving the demand for high-performance data analysis. Compared to other machine-learning algorithms, deep neural networks (DNNs) have achieved dramatic accuracy improvements over the past decade. They have now been employed in a vast range of application domains, from image classification [137] and object detection [90] to autonomous driving [19] and drone navigation [42]. Two classes of DNN—convolutional and recurrent (CNNs and RNNs)—are particularly popular. While CNNs excel in learning spatial features, RNNs are more suited to problems involving time series.

As tasks increase in complexity, inference architectures become deeper and more computationally expensive. For example, a small LeNet-5 model targetting the simple MNIST handwritten digit-classification task requires 680kop/cl (thousand arithmetic operations per classification, where an arithmetic operation is either an addition or multiplication), while a VGG16 implementation executing the 1000-class ImageNet task requires 31Gop/cl along with 550MiB of 32-bit floating-point weight storage [136]. The development of algorithms for reducing the computational and storage costs of DNN inference is therefore essential for throughput-, latency-, and energy-critical applications. Recent work has shown that, with the use of approximation, DNN deployment becomes more feasible thanks to its resultant reductions in memory use and compute complexity.

DNN approximation algorithms can be classified into two broad categories: quantisation and weight reduction. Quantisation methods reduce the precision of weights, activations (neuron outputs), or both, while weight reduction removes redundant parameters through pruning and structural simplification. By doing so, the latter commonly leads to reductions in numbers of activations per network as well. We assess methods of both types in this article, since they both contribute to DNN acceleration.

For many years, general-purpose processors (GPPs), particularly multi-core CPUs and GPUs, have been the dominant hardware platforms for DNN inference. For uncompressed DNN models, layer operations are mapped to dense floating-point matrix multiplications, which can be efficiently processed in parallel by GPPs following the single-instruction, multiple-data (SIMD) or single-instruction, multiple-thread (SIMT) parallel-processing paradigms. With DNN approximation, however, there is an emerging trend of using custom hardware platforms, such as field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs), to accelerate inference instead. While GPUs still excel at dense floating-point computation, researchers have reported higher throughput and energy efficiency with custom hardware through the use of low-precision fixed-point quantisation [66, 127]. Moreover, SIMD and SIMT architectures often perform poorly when operating on sparse data; DNNs compressed via fine-grained weight reduction have been shown to execute more efficiently in custom hardware [52, 109]. Logic and memory hierarchy customisability often make custom hardware DNN inference faster and significantly more energy efficient than through the use of GPPs.

A significant number of world-leading information technology firms have selected custom hardware over GPPs for the implementation of their next-generation DNN architectures. These include ASICs, e.g., Google's Tensor Processing Unit (TPU) [65], Intel Nervana [1], and IBM TrueNorth [2], as well as FPGA-based designs such as Microsoft Brainwave [28] and Xilinx Everest [151]. In general, ASIC designs can achieve state-of-the art throughput and energy efficiency. Their time-consuming and resource-demanding design and fabrication processes, however, make it hard for them to keep up with the rapid evolution of DNN algorithms [28, 66].

High-level implementation tools, including Intel's OpenCL Software Development Kit and Xilinx Vivado High-Level Synthesis, and Python-to-netlist neural network frameworks, such as

DNNWeaver [125], make the DNN hardware design process for both FPGAs and ASICs faster and simpler. Such software allows DNN architects unfamiliar with hardware development to migrate their designs to custom hardware with relative ease. Reconfigurability, meanwhile, enables rapid design iteration, making FPGAs ideal prototyping and deployment devices for cutting-edge DNNs.

Through this survey, we aim to equip researchers new to the field with a comprehensive grounding of DNN approximation, revealing how custom hardware is able to achieve greater performance than GPPs for inference. More specifically, we make the following novel contributions:

- We motivate DNN approximation for custom hardware by comparing the so-called *roofline models* [107] of comparable FPGA, ASIC, CPU, and GPU platforms of different scales.
- We survey key trends in approximation for state-of-the-art DNNs. We detail low-precision quantisation and weight-reduction methods, introducing recent algorithmic developments and assessing their relative strengths and weaknesses.
- We evaluate the performance of custom hardware implementations of each method, focussing on accuracy, compression, throughput, latency, and energy efficiency.
- Based on identified trends, we propose several promising directions for future research.

There are some existing surveys on DNN approximation. Cheng et al. [25], Guo et al. [49], Cheng et al. [24], and Sze et al. [136] surveyed algorithms for DNN compression and acceleration. Of these, Cheng et al. [24] briefly evaluated system-level designs for FPGA implementation. Guo et al. only surveyed quantisation methods; weight reduction was not mentioned. Nurvitadhi et al. compared Intel FPGA performance to that of GPU platforms for CNN inference benchmarks [104]. This article represents the first survey that provides not only a comprehensive evaluation of approximation algorithms for efficient DNN inference but also in-depth analysis and comparison of these algorithms' implementations in custom hardware, covering both CNNs and RNNs.

2 PERFORMANCE EVALUATION METRICS

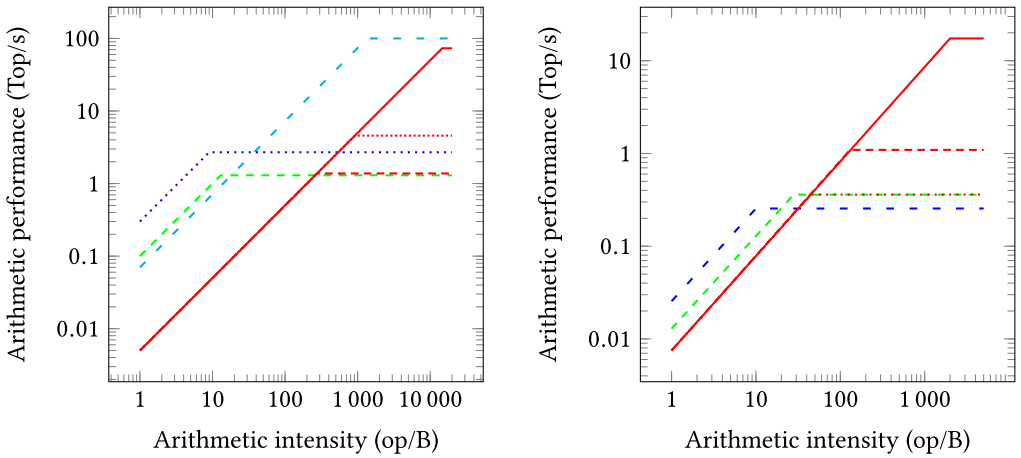
We evaluate the effectiveness of DNN approximation by considering the following factors.

- *Accuracy.* The two accuracy metrics commonly used in machine-learning research are *training* and *testing* accuracy, which, respectively, capture the proportions of correct classifications over training and testing datasets. Throughout this article, "accuracy" always refers to testing accuracy, which is indicative of a particular DNN's generalisability. *Top-n* accuracy captures the proportion of testing data for which any of the n highest-probability predictions match the correct results. Accuracies are reported as percentages, with changes expressed in percentage points (pp). Where comparisons are drawn against baselines, these are uncompressed implementations of the same networks, trained and tested using identical datasets, with all data in IEEE-754 single-precision floating-point format (FP32).
- *Compression ratio.* A network's weight storage requirement vs. that of the above baseline.
- *Throughput.* Classifications produced per second (cl/s). Also known as *classification rate.*
- *Latency.* The end-to-end processing time for one classification, in seconds (s).
- *Energy efficiency.* The throughput obtained per unit power, expressed in cl/J.

We also discuss application-specific considerations, e.g., parameter tuning time and design flexibility.

3 WHY CUSTOM HARDWARE? A ROOFLINE MODEL ANALYSIS

For DNN inference, approximation contributes to increases in throughput in three ways: *increased parallelism*, *memory transfer reductions* and *workload reductions*. With the help of roofline



(a) Datacentre-scale platforms: 18-core Intel Haswell CPU (- - -), Nvidia Tesla K80 GPU (·····), Google TPU ASIC (– –), and Xilinx Kintex UltraScale KU115 FPGA with 16-bit (– – –), eight-bit (·····), and one-bit (——) fixed-point weights [66, 141].

(b) Embedded-scale platforms: Nvidia Jetson TX1 GPU (– –), TI Keystone II DSP (- - -), and Xilinx Zynq ZC706 FPGA with 16-bit (·····), eight-bit (– – –), and one-bit (——) fixed-point weights [58].

Fig. 1. Comparison of roofline models of datacentre- and embedded-scale DNN inference platforms.

modelling, we can explain each factor's contribution, revealing why custom hardware can squeeze more speedup from approximation than GPPs.

A roofline model captures the theoretical peak performance of an acceleration platform while also reflecting the effects of off-chip memory data transfers. For any high-performance computing engine, the peak *arithmetic performance*, expressed in op/s, is limited by two factors: memory bandwidth and the amount of available compute resources. In the context of DNN inference, memory bandwidth limits the rate at which activations can be read and written, as well as that at which parameters stored off-chip can be fetched. By compute resources, we mean on-chip parallel-processing units able to perform operations: chiefly, multiplication. When *memory bound*, the arithmetic performance of a platform does not scale with any increase in parallelism. At the *compute bound*, meanwhile, all available processing resources are saturated.

Figure 1 overlays the estimated rooflines of DNN inference accelerators on several hardware platforms. The abscissa shows the *arithmetic intensity* of DNN inference, while the ordinate indicates the peak attainable arithmetic performance. Arithmetic intensity, also commonly referred to as *operational intensity* or *compute-to-communication (CTC) ratio*, is expressed as the number of arithmetic operations performed per byte of off-chip memory traffic (op/B). Arithmetic performance is memory bound when the arithmetic intensity is to the left of the break point. When to the right, it is compute bound: resource limitations prevent further scaling.

For fairness, platforms were divided into datacentre and embedded scales and compared accordingly. For FPGA-based accelerators, compute bounds were approximated under the assumption that the cost per fixed-point multiply-accumulate (MAC) unit was 2.5 lookup tables (LUTs) for one-bit (binary), 40 LUTs for eight-bit, and eight LUTs and half a digital signal processing (DSP) block for 16-bit precision, as suggested by Umuroglu et al. [141]. Both weights and activations were quantised at the same precision. We assumed that both Xilinx FPGAs featured, the Kintex UltraScale KU115 and Zynq ZC706, had 4.8GiB/s of off-chip memory bandwidth, and that implementations on the two devices were clocked at 350 and 200MHz, respectively [141].

3.1 Compute Bound Flexibility

From Figure 1, we can observe that, due to their specialised support for floating-point arithmetic operations, GPUs can deliver the highest arithmetic performance for FP32 DNN inference. When moving from floating-point to lower-precision fixed-point data representations, however, custom hardware design flexibility facilitates the trading off of precision for increased performance. Being robust to reductions in precision, DNNs can take great advantage of this flexibility [32]. The ASIC implementation featured, the TPU, has the greatest compute bound—92Top/s—following which is the KU115 FPGA. Since FPGAs afford their users total post-fabrication architectural freedom, different compute bounds are reachable, dependent upon the chosen precision, for the same device. As a result, the KU115 has compute bounds of 1.0Top/s with 16-bit, 3.0Top/s for eight-bit and 50Top/s for one-bit fixed-point representations. Similarly, the embedded-scale ZC706 can reach 360Mop/s for 16-bit, 1.0Top/s for eight-bit and 17Top/s for binary. Compared with custom hardware platforms, GPPs have lower compute bounds, since their arithmetic units are designed to perform high-precision operations and are thus inefficient when operating on low-precision data.

3.2 Arithmetic Performance Increases from Network Compression

Reaching a platform's compute bound is only possible if the executing application is not limited by its memory. If it is, then, to achieve higher arithmetic performance, higher arithmetic intensity is required. With network compression in the form of precision reductions, less off-chip memory needs to be accessed per operation performed, hence higher arithmetic intensity—and subsequently performance, if the application is not compute bound—is achievable. Networks can also be compressed via weight reduction, which both saves memory and removes the need to perform the associated operations. This can also lead to increased arithmetic intensity and thus performance: a smaller network can use on-chip caching more efficiently, reducing, or even entirely eliminating, off-chip memory traffic [141]. Performance gains from network compression can be supported from observations from the roofline models, in which, when bounded by memory, an increase in arithmetic intensity means a rightward shift along a roofline, resulting in an increase in arithmetic performance. Although all hardware platforms can benefit from network compression, custom hardware implementations, featuring higher compute bounds than GPPs, stand to gain the most; GPPs hit their compute bounds earlier when arithmetic intensity increases.

3.3 Limitations

While roofline models can allow one to predict increases in arithmetic performance (in op/s) that will arise from increased parallelism and memory transfer reductions gained through approximation, they can capture the corresponding changes in throughput (in cl/s) to only a limited extent. To understand the throughput impacts of weight-reduction methods, we must consider an additional factor. Arithmetic performance and throughput are related by *workload* (op/cl): the number of arithmetic operations performed per classification. Since weight reduction removes unimportant parameters, these methods achieve simultaneous memory transfer and workload reductions. As memory transfer reductions can facilitate arithmetic performance increases, it is possible for throughput increases to outpace those in arithmetic performance realised through their employment. Quantisation methods, however, do not cause reductions in workload, since the numbers of operations performed per classification remain the same. For these, increases in arithmetic performance result in proportional increases in throughput.

Roofline modelling does not account for the discrepancies in accuracy that arise from approximation. In general, while DNN approximation results in information loss and subsequent accuracy degradation, the majority of works surveyed in this article suggest that the acceptance of low to

moderate sacrifices in accuracy can result in significant performance improvement. Some show that, in certain scenarios, the introduction of approximation can actually improve accuracy by reducing model overfitting. The remainder of this article places great emphasis on the analysis of tradeoffs between network compression and accuracy.

Latency-critical DNN applications, such as advanced driver assistance systems, require the swift production of classifications. Many user-interfacing applications also require low latency to maintain adequate user experience [121]. Roofline models do not inherently capture latency. Herein, we detail how custom hardware can achieve state-of-the-art DNN inference latency, as well as throughput, thanks to its flexibility.

Approximation in custom hardware can also achieve superior energy efficiency—another metric whose behaviour is not natively observable through roofline modelling—vs. competing platforms. Custom hardware-based DNN inferencing applications operate at lower clock frequencies and hence consume less power, while also attaining higher throughput and/or lower latency, than those running on GPPs. Furthermore, some implementations, by exploiting customisability, outperform GPU-based versions in terms of memory energy efficiency.

4 QUANTISATION

The first major approximation theme we consider is that of quantisation. FPGA and ASIC flexibility permits the implementation of low-precision DNNs, thereby increasing throughput through parallelisation and by reducing reliance on slow off-chip memory.

4.1 Fixed-point Representation

4.1.1 Algorithmic Development. A floating-point-quantised DNN typically allows for an arbitrary binary point position, i.e., exponent value, for each individual parameter. This flexibility in data representation range comes at the expense of high resource use, power consumption, and arithmetic operation latency, however. Fixed-point-quantised DNNs generally use consistent, predetermined precisions and binary point locations, i.e., equal maximum and minimum representable magnitudes, for entire networks. This allows for fast, cheap, and power-efficient arithmetic operations in hardware, but enforces the use of constant data representation ranges. Early works, such as Courbariaux et al.'s [32], surveyed this topic, signalling that the accuracy of CNN inference can be preserved even with forward propagation conducted in low-precision fixed-point formats. Jacob et al. performed eight-bit quantisation of a popular CNN model, MobileNet, reporting an up-to 50% reduction in inference latency on an ARM CPU with only a 1.8pp accuracy drop for the Common Objects in Context (COCO) dataset [62]. Thereafter, many authors presented FPGA-based CNN and RNN inference frameworks using low-precision fixed-point formats that achieved superior throughputs to their floating-point counterparts with negligible accuracy drops [94, 155]. However, since data in different layers can have very different ranges, using a constant quantisation resolution for an entire network can provide suboptimal bandwidth efficiency.

Courbariaux et al. [32], Qiu et al. [111], and Shin et al. [129] explored using *block floating point (BFP)* for weight and activation quantisation. With BFP, often unfortunately referred to as "dynamic fixed point" [148], groups of variables share common binary point locations represented as scaling factors updated during training based on data distributions. As such, it can be seen as a compromise between fully floating- and fixed-point formats. These authors associated each layer's parameters with a scaling factor, updated after each arithmetic operation by checking the parameters' overflow status during training. Their experiments showed that, for both CNNs and RNNs, BFP quantisation of both weights and activations can result in the incursion of below 1.0pp accuracy losses. Since then, BFP has become common in the hardware inference of DNNs as well.

Many authors have explored methods allowing for the automatic selection of layer-wise precision. Inspired by Sung et al. [135], Shin et al. proposed the exhaustive search for cost-optimal precisions to use within long short-term memories (LSTMs) through analysis of the tradeoff between signal-to-quantisation-noise ratio (SQNR) and precision [129]. The time complexity of such searches is too high to be practical, however. Qiu et al. formulated an optimisation problem for minimising quantisation error with respect to changes in precision and binary point location [111]. A greedy method was proposed for its solution, resulting in desirable layer-wise CNN quantisations. Lin et al. [84] formulated and solved an SQNR-based optimisation problem to identify the optimal fixed-point precision per layer of a custom-designed CNN, showing that the proposed scheme offered over 1.2× compression for the CIFAR-10 dataset with no loss in accuracy. Their method converts pretrained networks from FP32 into further-quantised equivalents without retraining.

Many authors have focussed on reducing accuracy losses through the modification of rounding schemes. Gupta et al. trained CNNs with 16-bit fixed-point weight representation using stochastic rounding, achieving lossless compression for the MNIST and CIFAR-10 datasets [51]. By following

$$\text{round}(x) = \begin{cases} \lfloor x \rfloor & \text{with probability } 1 - \frac{x - \lfloor x \rfloor}{2^{-f}} \\ \lfloor x \rfloor + 2^{-f} & \text{otherwise,} \end{cases} \tag{1}$$

stochastic rounding results in input x being rounded with resolution 2^{-f}, where f is the fractional width of the result. The probability of rounding x to $\lfloor x \rfloor$ is proportional to the proximity of x to $\lfloor x \rfloor$. Stochastic rounding is thus an unbiased scheme, i.e., $\mathrm{E}\left[\text{round}(x)\right] = x$. Wu et al. proposed WAGE, a CNN framework that discretises gradients using stochastic rounding [150]. Using two bits for weights and eight bits for activations, gradients, and errors, AlexNet trained to classify ImageNet with WAGE exhibited an around-8.8pp drop in accuracy. Shin et al. explored treating quantisation resolution as a trainable parameter for both CNNs and RNNs [128]. With a tunable quantisation granularity, a four-bit CNN classifying the SVHN dataset and a six-bit RNN performing language modelling each achieved less than 0.1pp of accuracy loss.

While all of the previously mentioned works featured weights quantised using fixed-point formats, Lai et al. implemented CNN inferencing with floating-point weights and fixed-point activations [72]. Experiments with AlexNet showed that the use of seven-bit floating-point weights could achieve the same accuracy as 11-bit fixed-point representation with ImageNet. The authors suggested that weight range is more important than precision in preserving accuracy. This observation laid the foundations for logarithmic quantisation (Section 4.3), which trades off precision for range.

The authors of Adaptive Quantisation investigated quantisation at a finer granularity than the aforementioned down-to layer-wise methods [69]. During retraining, networks adapt, with each filter allowed to assume an independent precision. Experiments with small-scale datasets and models showed that Adaptive Quantisation, when combined with pruning, is able to achieve accuracies and compression ratios superior to binarised neural networks, for which each datum is represented using only a single bit. A framework for implementing low-precision quantisation, DoReFa-Net, supports arbitrary precisions for weights, activations, and gradients, from 32-bit fixed point down to binary [161]. Its authors conducted empirical analysis of various data precision combinations, concluding that accuracy deteriorates rapidly when weights and/or activations are quantised to fewer than four bits.

4.1.2 Hardware Implementation. Nurvitadhi et al. conducted experiments to evaluate the performance of Nvidia GPUs and Intel FPGAs for CNN inference using floating- and fixed-point data representations [104]. They concluded that, while their evaluated Stratix-10 FPGA's throughput lagged a Titan X GPU's with FP32, the FPGA could enable over 50% greater throughput with six-bit

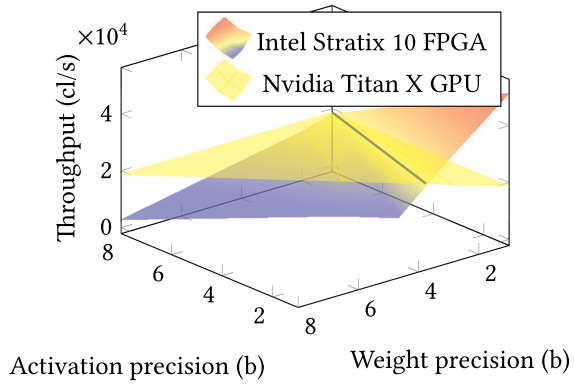


Fig. 2. Throughput comparison of Intel Stratix 10 FPGA and Nvidia Titan X GPU AlexNet implementations classifying the ImageNet dataset using various fixed-point weight and activation data representations [29].

fixed-point data. The throughput advantages and energy savings of FPGAs become more significant as precision decreases. Colangelo et al. presented an Intel FPGA-based inference framework taking advantage of bandwidth and computation savings from low-precision data [29]. Their experimental results for AlexNet, as presented in Figure 2, showed that, as precision fell, the throughput of their FPGA implementation improved and eventually exceeded that of a GPU of similar scale, supporting the conclusions by Nurvitadhi et al. The FPGA achieved an order-of-magnitude throughput improvement over the GPU at binary precision. Zhang et al. showed that a fixed-point-quantised long-term recurrent convolutional network (LRCN) implementation on a Xilinx Virtex 7 VC709 FPGA could achieve a 3.1× throughput speedup vs. an Nvidia K80 GPU equivalent [157].

Köster et al. presented Flexpoint, another BFP variant, for CNN training and inference [70]. Using the "flex16+5" (16-bit mantissa and five-bit shared exponent) data format, Intel's neural network ASIC, Nervana, was shown to achieve the same accuracy as FP32, while reducing memory bandwidth by around 50%, for the training of AlexNet and ResNet with ImageNet.

The latest-generation Intel FPGAs can pack up to either one 27-×27-bit or two 18×19 MAC(s) per DSP block. When using lower precisions on FPGAs, many authors have implemented multipliers using LUTs instead of DSPs to achieve higher resource efficiency. Boutros et al. proposed the enhancement of DSP blocks to support low-precision MACs with some 12% area overhead and no drop in achievable frequency [15]. One such enhanced DSP can perform one 27×27 or two 18×19, four 9×9 or eight 4×4 parallel MAC(s). The authors implemented AlexNet, VGG-16, and ResNet-50 using the enhanced DSPs. On average, they improved the throughput of eight-bit and four-bit DNNs by 1.3× and 1.6×, respectively, while correspondingly reducing the occupied area by 15% and 30% compared to the default use of DSPs in the Intel Arria 10 they targetted.

Sharma et al. [126] and Moons et al. [99] both introduced variable-precision bit-parallel ASIC implementations. Sharma et al.'s Bit Fusion consists of an array of bit-level MACs that dynamically fuse to match the precisions of individual DNN layers [126]. Experiments with AlexNet showed that Bit Fusion, while consuming only 900mW of power, is only 16% slower than an Nvidia Titan Xp implementation using its native eight-bit vector instructions. The Titan Xp can consume up to 250W of power. Moons et al. used similar ideas, with their implementation consuming 76mW to achieve 47cl/s for AlexNet, outperforming static-precision Eyeriss by 3.9× in energy efficiency [99].

Having realised the importance of flexibility of precision in achieving high DNN inference efficiency, GPP manufacturers have recently begun to offer support for low-precision MACs. Intel Cascade Lake CPUs provide so-called Vector Neural Network Instructions in 16- and 8-bit

formats [61], while Nvidia Turing GPUs support TensorRT, a deep-learning platform integrable with TensorFlow, allowing for low-precision arithmetic down to as few as four bits [106].

We can categorise MACs into two families: bit-parallel and -serial. FPGA- and ASIC-based DNN inference architectures with consistent precision generally use bit-parallel MACs for performance and/or simplicity of reuse. fpgaConvNet [142], Angel-Eye [48], ESE [52], and works by Chang et al. [18] and Shen et al. [127] represent the state-of-the-art in FPGA-based CNN and RNN implementation using low-precision bit-parallel MACs. DaDianNao [22], Cnvlutin [4], NeuFlow [40], and the TPU [66], meanwhile, are cutting-edge ASIC-based bit-parallel DNN inference platforms. For bit-parallel MACs, DNN hardware is typically designed to natively support the maximum precision of an entire network. However, as suggested by Khoram et al. [69] and Li et al. [82], since actual precision requirements vary considerably across DNN layers, bit-parallel DNN hardware typically processes an excess of bits per operation. Bit-serial alternatives, however, allow precision to be trivially varied at runtime, making their use suitable for fine-grained mixed-precision networks.

Stripes [67], Loom [123], and Bit Pragmatic (PRA) [3] are ASIC-based DNN accelerators that perform layer-wise mixed-precision inference using bit-serial MACs. Among these, experiments showed that Stripes achieved a 1.3× throughput increase over bit-parallel DaDianNao with VGG-19 [67]. Based on Stripes, Albericio et al. proposed an ASIC implementation, PRA, which performs bit-serial neuron activations by shifting inputs with respect to the indices of non-zero bits in the weights [3]. Experiments showed that PRA could achieve 2.6× and 2.0× increases in throughput and energy efficiency, respectively, vs. DaDianNao. Gudovskiy et al. proposed an FPGA implementation, ShiftCNN, using similar ideas to PRA [47]. ShiftCNN was shown to obtain 4.2× and 3.8× energy efficiency savings over two baseline CNN platforms using DSP- and LUT-based bit-parallel MACs, respectively. Moss et al. presented an FPGA-based customisable matrix multiplication framework dedicated to DNN inference [100]. Their implementation allows for the runtime switching between static-precision bit-parallel and dynamic-precision bit-serial MAC implementations. They observed up-to 50× throughput increases vs. FP32 baselines for AlexNet, VGGNet and ResNet.

4.2 Binarisation and Ternarisation

4.2.1 Algorithmic Development. *Binarisation* is the quantisation of parameters into just two values, typically $\{-1, 1\}$ with a scaling factor. Although binary quantisation leads to the incursion of greater error than non-binary fixed-point quantisation, inference operations can be substantially simplified. Early works, such as BinaryConnect, focussed on *partial* binarisation, for which only weights are binarised [31]. *Full* binarisation of CNNs was proposed in BinaryNet: both weights and activations are binarised [30]. For binarised training, weights are binarised only during forward propagation; they are not binarised during backward propagation, since stochastic gradient descent is sensitive to quantisation and does not work well with very low precisions.

The authors of BinaryConnect and BinaryNet proposed binarisation of two types: deterministic and stochastic. For deterministic binarisation, a simple sign function is used, while the stochastic binarisation process is equivalent to stochastic rounding, as was shown in Equation (1). Since the derivative of the sign function is a Dirac delta function with zero everywhere but the origin, rendering the training process impossible, the authors of BinaryNet resorted to using a hard hyperbolic tangent (tanh) function to cope with this problem during backward propagation [30]:

$$\tanh_{\text{hard}}(x) = \begin{cases} 1 & \text{if } x > 1 \\ x & \text{if } -1 \leq x \leq 1 \\ -1 & \text{otherwise.} \end{cases}$$

In this way, the gradient of their cost function could be preserved for weights within $[-1, 1]$ during training. Clipping was also applied to the real-valued weights to constrain them to $[-1, 1]$. Experiments with the MNIST and CIFAR-10 datasets on unidentified networks showed that BinaryConnect achieved around-1–2pp higher prediction accuracies than FP32 baselines. The authors suggested that this was due to stochastic rounding's regularisation effect, whence randomisation is injected into a network in a similar way to "dropout" in the form of per-neuron binarisation noise [133]. Experiments with BinaryNet with MNIST, CIFAR-10 and SVHN—also on unknown networks—showed less-than 1pp accuracy losses compared to baseline cases. However, this regularisation effect was only seen for small datasets. For large-scale ones such as ImageNet, although BinaryNet with AlexNet achieved significant memory and computational complexity reductions, this was accompanied by around-30pp top-one accuracy drops. Binarisation's high error inducement outweighed the positives of regularisation in these cases.

In an effort to improve BinaryNet's data representation, XNOR-Net features trainable filter-wise scaling factors for forward propagation [112]. These scaling factors retain the average magnitudes of weights and activations to improve the expressiveness of binarised networks. Experiments with XNOR-Net inferencing AlexNet with the ImageNet dataset showed that this method successfully improved top-one accuracy by around 20pp compared with BinaryNet, while there was still an accuracy drop of over 10pp vs. an FP32 baseline. XNOR-Net does, however, require averaging operations over input features, adding costly high-precision dividers [44].

ABC-Net alleviates the information loss from binarisation by approximating FP32 parameters and activations as linear combinations of multiple binary values [86]. Its authors pointed out that, during forward propagation, their K-binarisation scheme (K parallel bitwise XNORs) is cheaper than performing K-bit fixed-point multiplication, emphasising ABC-Net's superior resource efficiency over conventional fixed-point CNN implementations. A five-bit weight/activation ABC-Net achieved a 14pp top-one accuracy improvement vs. XNOR-Net with ImageNet on ResNet-18.

Tang et al. proposed a number of improvements to the binarised retraining process [139]. One of their discoveries was that a low learning rate is preferable to avoid frequent parameter oscillation, which leads to prolonged and inefficient training. Furthermore, a binary-constrained regulariser was added to their training loss function to encourage more bipolar weight values (closer to ±1). This was implemented within the function as

$$\text{loss}_{\text{post-reg}}(\boldsymbol{W}, \boldsymbol{b}) = \text{loss}_{\text{task}}(\boldsymbol{W}, \boldsymbol{b}) + \lambda \sum_{l=1}^{L} \sum_{n=1}^{N_l} \sum_{m=1}^{M_l} 1 - \boldsymbol{W}_{lnm}^2, \tag{2}$$

wherein $\boldsymbol{W}$, $\boldsymbol{b}$, and λ represent weight, bias and regularisation factor, respectively. L is the network's depth and M_l and N_l are the input and output channel numbers in the lth layer. $\text{loss}_{\text{task}}(\boldsymbol{W}, \boldsymbol{b})$ returns the task-related loss based on the original network settings, while $\text{loss}_{\text{post-reg}}(\boldsymbol{W}, \boldsymbol{b})$ gives the post-regularisation loss. Tang et al.'s regulariser penalised with respect to the implemented network's overall quantisation loss. These optimisations, together with multi-bit activation representation, resulted in a 6.4pp top-one AlexNet accuracy increase over XNOR-Net for ImageNet.

Going further, HWGQ addressed the problem of mismatching gradients between the binarised forward activation function, sign, and the backward activation function, hard tanh [16]. HWGQ uses a half-wave Gaussian-quantised (HWGQ) rectified linear unit (ReLU) for forward propagation and a standard ReLU function for backward propagation. The authors' experiments with AlexNet produced a 47% top-one ImageNet error rate: the lowest achieved for a binary network to date.

Ott et al. suggested that RNNs are not amenable to binarisation, since the large quantisation losses of near-zero values forced to ±1 get amplified over their recursions [108]. Nevertheless, Liu et al. implemented binarisation in LSTMs targetting English and Chinese language modelling, although they only applied it to input and output embedding layers (those that encode text as vectors) [91]. The authors reported up to 11× compression of those layers without accuracy loss. Given these seemingly conflicting conclusions, further experiments are required to establish the effectiveness of binarisation in RNNs.

Adding zero to the binary value set gives *ternary* representation. TernaryConnect [87] and Ott et al.'s work [108] introduced ternary CNNs and RNNs, respectively, for improved accuracy. The accuracies of TernaryConnect exceeded the previous-best results for MNIST, CIFAR-10, and SVHN reported by the authors of BinaryConnect [31]. For each layer l, Ternary Weight Networks (TWNs) use tunable symmetric thresholds $\pm\delta_l$ to differentiate 0 from ±1 [78]. For an AlexNet implementation classifying ImageNet, TWNs achieved a 46% top-one error rate: lower than all binarised neural networks reported thus far. In Trained Ternary Quantization, parameters are represented in the form $\{w_l^-, 0, w_l^+\}$, wherein w_l^- and w_l^+ are trainable [162]. Compared with TWNs, a further accuracy improvement—around 5pp—was reported for AlexNet with ImageNet.

Mellempudi et al. presented Fine-grained Quantisation (FGQ), which involves the ternarisation of a pretrained FP32 network into groups, then ternarising each group independently [95]. Within a group g, the ternary weights can have distinct quantisation levels $\{-w_g, 0, w_g\}$. Although groups can be determined arbitrarily, in this case the authors grouped by channel to promote implementational efficiency. Assuming that a network has G such groups, there are $2G + 1$ distinct levels with which to represent weights in total, increasing the model's representation capacity over ternarisation with equal granularity. Weights are partitioned along channels for simplicity. Experiments with ImageNet showed that an FGQ-quantised AlexNet with ternary weights and four-bit activations suffered 7.8pp accuracy loss compared to the baseline.

Alemdar et al. combined ternarisation with knowledge distillation, in which shallower "student" networks are used to mimic deeper "teachers" [5]. In hardware, ternarisation requires cheaper arithmetic operators than higher-than-two-bit fixed-point quantisation. To improve the accuracy of a ternary student network, stochastic rounding (Equation (1)) is used while ternarising during teacher network backward propagation. Experiments with MNIST, CIFAR-10, and SVHN on arbitrarily chosen models showed that ASIC implementations of this work achieved 3.1× greater energy efficiency, on average, than IBM TrueNorth executing the same benchmarks with ternary data [2].

While low-precision networks lead to significant network compression, they often require higher numbers of neurons to achieve accuracies comparable to their floating-point counterparts. For the CIFAR-10 dataset, for example, binary networks such as FINN [141] and ReBNet [44] require a wider and deeper model, CNV, to achieve similar accuracy to an FP32 baseline with CifarNet, a much thinner and shallower model [130]. Zhu et al. proposed the Binary Ensemble Neural Network (BENN), in which multiple binarised networks are aggregated by "boosting" (parallel ensemble with trained weights) [163]. The authors showed that their network ensembles exhibited lower bias and variance than their individual constituents while also having improved robustness to noise. Experiments with AlexNet on the ImageNet dataset showed that the use of BENN, with AdaBoost (adaptive boosting) and an ensemble of six binarised networks, led to only 2.3pp of top-one accuracy loss vs. an FP32 baseline. The authors of WRPN explored the same phenomenon by gradually reducing network precision and increasing the number of channels of an originally FP32 network, finding that, by increasing model complexity, a low-precision network can eventually match or even surpass the accuracy of its baseline. Further research is required to identify models that are particularly amenable to low-precision inference [96].

4.2.2 Hardware Implementation. For inference, binary networks have several properties that enable elegant mapping to Boolean operations. With a set bit representing 1 and an unset bit −1, multiplication becomes an XNOR operation: significantly cheaper to implement than non-binary fixed-point multiplication. Furthermore, accumulation becomes a population count (popcount) operation, which, on an FPGA, requires half the LUTs of an equivalent adder tree [141]. Umuroglu et al. [141] and Ghasemzadeh et al. [44] suggested that, during binary inference, operations in batch normalisation can be simplified to binary thresholding, where $y = \text{sign}(\alpha x - b) = \text{sign}(x - {}^{b}/_{\alpha})$. x, α, b, and y are the input, scaling factor, bias, and output, respectively. A max-, min-, or average-pooling layer in a binary network can be efficiently implemented using OR, AND, or majority functions.

On GPUs, 32 one-bit activations and weights can be packed into each word to perform bit-wise XNORs. On a Titan X Pascal GPU, 32 32-bit popcounts can be issued per cycle per streaming multiprocessor (SM). Thus, up to 512 binary MAC operations can be performed per cycle per SM. As it can issue up to 128 FP32 MAC instructions per cycle per SM, however, it can be estimated that the theoretical peak throughput gain of a binary network over FP32 for that GPU is only 4× [104].

On FPGAs, binary network inference can show more significant performance gains. Many frameworks, including FINN [141], FP-BNN [83], and that from Moss et al. [100], have been built to achieve this, resulting in orders of magnitude higher throughput and energy efficiency than floating-point counterparts of comparable scale. FINN's authors constructed small binary networks for the MNIST, CIFAR-10, and SVHN datasets targetting the Xilinx Zynq ZC706 FPGA. Experiments with the CNV network (110Mop/cl) resulted in sustained throughput of 22kcl/s—the highest throughput at the time of publication—while consuming as little as 25W of power. The authors of FP-BNN implemented AlexNet (2.3Gop/cl), one of the larger CNNs, on an Intel Stratix V FPGA, reporting a throughput of 870cl/s, 2.7× faster than a 235W-consuming Tesla K40 GPU executing the same binary network, while drawing only 26W of power. On a smaller custom network designed for CIFAR-10 inference (1.2Gop/cl), in which arithmetic intensity was higher, FP-BNN achieved a peak throughput of 7.6kcl/s. Moss et al. showed that, with binarisation, the HARPv2 heterogeneous platform could achieve a peak throughput of 110cl/s for VGGNet, with 1.2× greater energy efficiency than a Titan X Pascal GPU-based alternative [100].

The authors of ReBNet implemented "residual binarisation" on FPGAs [44]: similar to ABC-NET's aforementioned K-binarisation scheme [86]. They observed accuracy improvements when higher data widths were used, as was the case for ABC-Net. ReBNet's authors reported that their work exposes a continuum between accuracy and area, making it amenable to a wide range of application requirements and hardware constraints.

Prost-Boucle et al. implemented ternary CNNs on a Xilinx Virtex-7 VC709 FPGA, presenting both high-performance- and low-power-targetting designs [110]. Their experiments with the CNV model classifying CIFAR-10 demonstrated a 6.6pp accuracy improvement compared to FINN's binarised inference. In high-performance mode, up to 27kcl/s was achieved with around 13W of power consumption while, in low-power mode, 14kcl/s was obtained for half the power.

The authors of YodaNN introduced a 65nm ASIC implementation featuring partial binarisation, in which activations and weights are quantised to 12 and one bit(s), respectively [9]. Experiments with AlexNet and the ImageNet dataset showed that YodaNN achieved a throughput of 0.50cl/s and an energy efficiency of 2.0kcl/J at 0.60V.

4.3 Logarithmic Quantisation

4.3.1 Algorithmic Development. In a base-two logarithmic representation, parameters are quantised into powers of two with a scaling factor. Suiting the observation that a weight's representation range is more important than its precision in preserving network accuracy, logarithmic

representations can cover wide ranges using few bits [72]. While logarithmic representation can also be used for activations, this has yet to be explored. LogNet's authors quantised CNNs with weights encoded in a four-bit logarithmic format, after which they performed retraining to recover some lost accuracy [76]. Their experiments with the ImageNet dataset revealed 4.9pp and 4.6pp top-five accuracy drops for AlexNet and VGG16, respectively. In Incremental Quantisation (INQ), weights are iteratively quantised into a logarithmic format, with activations left as eight-bit fixed point values [159]. In each iteration, parameters in each layer are partitioned into two groups using a threshold on absolute parameter values. The group with higher absolute values is quantised into powers of two directly, whereas the other is retrained in the following iteration in FP32 to compensate for losses. This process repeats until all parameters are quantised. Experiments with ImageNet on AlexNet showed a negligible (~0.1 pp) accuracy loss against the baseline while using only five bits per weight.

4.3.2 Hardware Implementation. For hardware inference, base-two logarithmic representations see multiplications converted into binary shifts for greater area and energy efficiencies as well as speed. GPPs perform binary shifts using shifters embedded in arithmetic and logic units, most of which can move their operands by an arbitrary number of bits per operation. On an Nvidia Maxwell GPU, the theoretical peak throughput of 32-bit binary shifts is 50% of that of FP32 MACs [105].

In custom hardware, a multiplication between an exponentially quantised weight parameter and an activation can be implemented cheaply using a variable-length binary shifter. With LogNet, CNN inference is performed on FPGAs with four-bit logarithmic-quantised weights [76]. Experiments with three convolutional layers showed an over-3.0× energy efficiency improvement vs. an Nvidia Titan X GPU implementation, while a four-bit logarithmic implementation of AlexNet demonstrated an around-5pp accuracy loss for ImageNet. Wang et al. implemented base-two logarithmic quantisation on weights associated with input, output and forget gates in LSTMs while leaving the remaining gates in non-logarithmic eight-bit fixed-point precision [146]. In their 90nm ASIC implementation, multiplications with logarithmic-quantised weights are implemented with shift-and-add operations, which occupy significantly less area than MACs using non-logarithmic fixed-point quantisation. Wang et al.'s ASIC was able to process a 512×512 LSTM layer within 1.7μs at a silicon area cost of 31mm^2.

The implementations mentioned above reuse binary shifters over different groups of weights for scalability. For custom hardware, if shift amounts are constant, no logic is required for multiplication: they can be performed in routing alone. This means that fixing DNN parameters using constant-length shifts instead of multiplications can result in significant resource and latency savings. Server-scale platforms with massive resource availability, such as Microsoft Catapult [17] and Amazon Web Services, should be able to benefit hugely from such optimisations.

5 WEIGHT REDUCTION

Let us now turn to DNN approximation's second key subject: weight reduction. Here, parameters deemed unimportant are eliminated entirely. Weight reduction improves the performance of hardware inference by reducing both workload and off-chip memory traffic.

5.1 Pruning

5.1.1 Algorithmic Development. Pruning is the process of removing redundant connections in a DNN. Inspired by early works including Optimal Brain Damage [75] and Optimal Brain Surgeon [56], Srinivas et al. proposed a retraining-free method for removing redundant neurons in trained CNNs [132]. Similar neurons can be wired together and hence pruned away. The authors proposed the similarity evaluation of neurons using a matrix of their squared Euclidean distances.

This method resulted in 6.7× and 1.5× compression for the multilayer perceptron and AlexNet networks, respectively. Experiments with AlexNet revealed 2.2pp of ImageNet accuracy loss.

Han et al. were the first to propose an iterative pruning process [55]. In their work, one iteration consists of pruning followed by retraining, allowing the remaining connections to learn to compensate for the pruning loss. After many such iterations, lossless compression ratios of 9.0 and 13 were achieved for AlexNet and VGG16, respectively, both classifying the ImageNet dataset. The authors attempted to promote sparsity in the networks by penalising non-zero parameters with an l_1 or l_2 norm-based sparsity regulariser [34] during retraining. An l_2 norm-based sparsity regulariser can be implemented as

$$\text{loss}'_{\text{post-reg}}(\boldsymbol{W}, \boldsymbol{b}) = \text{loss}_{\text{task}}(\boldsymbol{W}, \boldsymbol{b}) + \lambda \sqrt{\sum_{l=1}^{L} \sum_{n=1}^{N_l} \sum_{m=1}^{M_l} \boldsymbol{W}_{lnm}^2}, \tag{3}$$

wherein $\boldsymbol{W}$, $\boldsymbol{b}$, λ, L, M_l, N_l and $\text{loss}_{\text{task}}(\boldsymbol{W}, \boldsymbol{b})$ share Equation (2)'s definitions, while $\text{loss}'_{\text{post-reg}}(\boldsymbol{W}, \boldsymbol{b})$ gives the post-regularisation loss. During training, this regulariser penalises Han et al.'s loss function with respect to the magnitudes of non-zero weights, resulting in more weights near zero. See et al. implemented a similar strategy for RNNs, finding that their 5.0×-compressed network for neural machine translation actually surpassed the baseline's accuracy for the WMT'14 dataset due to the effect of regularisation [122].

Following the idea of incorporating sparsity into training objective functions, Zhou et al. implemented low-rank constraints [88, 160]. The authors aimed to induce lower average ranks in weight matrices using a group sparsity constraint with a regulariser of the form of Equation (3). They achieved an AlexNet compression ratio of 4.3, inducing 0.57pp of top-one ImageNet accuracy loss.

Inspired by Han et al.'s work, the authors of Dynamic Network Surgery (DNS) performed pruning followed by "splicing," wherein the salience (importance) of the remaining parameters is evaluated; parameters' salience varies when others are removed [50]. DNS achieved 110× and 18× compression for LeNet-5 and AlexNet, respectively.

The proposals above all see DNNs pruned at element-wise granularity, often referred to as *fine-grained* pruning. Although pruning at the finest granularity leads to excellent compression ratios, it can also result in significant irregularities in weight distribution, which, in turn, can make it difficult for the inference hardware to convert compression into increased throughput. *Coarse-grained* pruning methods have hence been proposed, which produce larger but denser networks than those resulting from fine-trained pruning. Lebedev et al. introduced Structured Brain Damage, wherein a group-wise sparsification regulariser (Equation (3)) shapes each weight matrix's non-zeroes into a regular, dense pattern [74]. Experiments showed 3.0× improvements in both compression ratio and throughput with sub-1.5pp accuracy degradation for AlexNet classifying ImageNet. Wen et al. [147], Li et al. [80], He et al. [57], and Su et al. [134] performed structured pruning along channels, filters, layers, and shapes (arbitrary groups of parameters) of CNNs. All of these works proposed the pruning of groups of redundant parameters based on sums of parameter magnitudes where, intuitively, those with lower values are deemed less important.

The authors of Network Slimming argued that, although sparsity can be realised at different granularities, pruning at the channel level provides a tradeoff between flexibility and ease of hardware implementation [92]. The output of Network Slimming is simply a "thinned" version of an unpruned network. With every convolutional and fully connected layer followed by a batch-normalisation layer, networks are trained before pruning such that batch normalisation scaling factors represent the relative importance of each channel. Layer-wise pruning is then performed by thresholding them. An l_1 sparsity regulariser is used on the scaling factors, instead of each

parameter, to promote channel-wise sparsity. 20× compression and 5× workload reduction were reported against an unpruned baseline for VGGNet. Experiments with ImageNet on the VGG-A model demonstrated 5.8× compression with less than 0.1pp of accuracy loss.

Decisions on whether to prune specific parameters are based on parameter salience. Establishing accurate salience estimations is thus crucial for pruning effectiveness. Molchanov et al. proposed and compared various criteria for determining weight salience, including pruning by the magnitude, mutual information (against classification ground truth), and Taylor expansion of quantisation noise [97]. Of these, the Taylor expansion-based criterion was found to perform particularly well. Unlike the works above, which all defined parameter salience as the impact on accuracy, Yang et al. defined it as the impact on energy efficiency, achieving an energy saving of 3.7× with ImageNet on AlexNet against an Nvidia Titan X GPU equivalent [152].

5.1.2 Hardware Implementation. Coarse-grained pruning produces outputs in structured and dense patterns such that the Basic Linear Algebra Subprograms (BLAS) for GPPs can directly benefit from reductions in workload. It is more challenging for GPPs to benefit from fine-grained pruning, however. Modern GPUs follow a SIMT execution model, in which threads execute the same sequence of instructions on different data. Compute speed is thus bottlenecked by the slowest thread; others remain idle until synchronisation points are reached. Checking for zeroes in matrices adds extra instructions to each thread, further reducing computational efficiency. An alternative approach is to use linear algebra libraries supporting zero-skipping, such as sparse matrix-vector multiplication (SPMV). Monakov et al. proposed a matrix storage format that improves locality and enables automatic parameter tuning on GPUs [98]. Bell et al. implemented data structures and algorithms for SPMV on an Nvidia GeForce GTX 280 GPU, with which they achieved state-of-the-art FP32 performance [12]. For SPMV to show performance and/or storage advantages, however, matrices need to be highly sparse. This is often the case for RNNs, which normally have over 80% sparsity [52], but is not usually true for CNNs (typically only 5–50% sparsity) [74].

Custom hardware can handle irregular, sparse data more efficiently than GPPs for fine-grained-pruned DNNs. Li et al. presented an FPGA design framework for CNN sparsification and acceleration [81]. Their work features a load balancing-aware sparsification training scheme facilitating efficient parallelism. Their FPGA implementation of AlexNet achieved 12× throughput acceleration over an Intel Xeon CPU-based benchmark. Posewsky et al. presented an FPGA implementation of high-throughput zero-skipping suiting fine-grained pruning [109]. The authors proposed that, post-pruning, each non-zero weight be encoded as a two-element tuple (w_i, z_i) containing weight value w_i and number of preceding zeroes z_i, where i is the weight's index. In this way, when a batch of input activations is buffered on-chip, the hardware will only fetch the weights pointed to by z_i, corresponding to non-zeroes only. Experiments with an unidentified model showed that their Xilinx Zynq XC7Z020 FPGA implementation surpassed the throughput of ARM Cortex-A9 and Intel Core i7-5600U CPU equivalents, with >85% energy savings.

ESE's authors reported that, with pruning and retraining, more than 90% of the parameters of an arbitrarily chosen LSTM trained on the TIMIT dataset could be pruned away without harming accuracy [52]. Its authors proposed "balance-aware" pruning to shape weight matrices into equal workloads for parallel compute units during retraining. On FPGAs, weight matrices are stored and computed in a compressed sparse column format to skip zeroes under this proposal. ESE demonstrated 3.0× throughput acceleration vs. an Nvidia Pascal Titan X GPU implementation.

The authors of Eyeriss [23], EIE [53], Cnvlutin [4], and Laconic [124] sought to remove multiplications by zero-valued activations. The authors of Cnvlutin achieved this by computing only non-zero inputs and using an "offset" buffer, alongside the input buffer, to store the indices of each

input's corresponding weights after zero-skipping. A hardware controller fills the offset buffer on the fly such that it does not consume extra bandwidth. To further increase acceleration, Cnvlutin prunes near-zero outputs during inference to increase the sparsity of the next layer's input buffer. Experiments with several CNNs, including AlexNet, GoogleNet, and VGG-19, showed 1.2–1.6× throughput increases over DaDianNao [22] without any loss in accuracy for ImageNet. While Cnvlutin incurred an area overhead of 4.5% over DaDianNao, it beat it by 1.5× in terms of energy efficiency for an unnamed model. Eyeriss, EIE, and Laconic's authors achieved benefits from pruning using similar strategies to those employed by Cnvlutin's.

Unlike the previous proposals, which all prune parameters to achieve throughput speedups, the authors of Eyeriss and Minerva targetted energy savings through the elimination of redundant off-chip memory fetches [23, 114]. Experiments with Minerva showed that their 40nm ASIC implementation achieved an 8.1× energy efficiency reduction—also for an unidentified model—compared with an ASIC baseline.

5.2 Weight Sharing

5.2.1 Algorithmic Development. Weight sharing groups parameters into buckets, reducing network size as well as enabling multiplications to be converted into cheaper table lookups. In HashedNets, a low-cost hash function is used to randomly group connection weights, the connections in each of which all share a single value [21]. These parameters are then trained to adjust to the weight sharing with standard backward propagation. Experiments with the MNIST dataset showed that HashedNets achieved a compression ratio of 64 with an around-0.7pp accuracy improvement against a five-layer CNN baseline. The authors suggested that the accuracy rise could be attributed to the "virtual" connections created that seemingly increased expressiveness.

Ullrich et al. performed retraining using soft weight sharing on pretrained networks to fine-tune the centroids used for parameter clustering [140]. Soft weight sharing was originally proposed by Nowlan and Hinton, who modelled cluster centroids with a mixture of Gaussians [102]. When retraining with this constraint, weights tend to concentrate very tightly around a number of cluster components, the centroids of which optimise to improve accuracy. Experiments showed 160× compression for MNIST on LeNet-5 with an accuracy loss of ~0.1pp.

With Deep Compression, weight sharing is performed in several steps [54]. A network is first pruned with iterative retraining [55], after which weights are quantised via k-means clustering. The quantised network is then retrained again to fine-tune the remaining connections and update the cluster centroids. Finally, the quantised weights are compressed with Huffman coding to save memory. With k-means clustering, the spatial complexity of a size-K weight matrix reduces from $O(K^2)$ to $O(k)$. Using their basket of approximation techniques, the authors of Deep Compression achieved 35× overall compression for AlexNet with no drop in ImageNet accuracy.

The proposals above only encode weights. Both LookNN [113] and Quantised CNN [149] follow the "product quantisation" algorithm [64], which encode both weights and activations. Rather than operating element-wise, this method does so on subvectors of weight matrices. Experiments with Quantised CNN revealed 19× AlexNet compression in return for 1.5pp of ImageNet accuracy loss.

5.2.2 Hardware Implementation. During inference, weight sharing-based implementations require a large number of lookup operations, which can be performed significantly more efficiently on FPGAs than GPPs. Samragh et al. implemented weight sharing on FPGAs [120]. Here, k-means cluster centroids are determined with tunable parameters during retraining, eliminating almost all multiplications. An up-to 15× improvement in throughput and compression ratio of 9.0 were reported along with with sub-0.1pp of accuracy loss for small DNN datasets such as MNIST and ISOLET on unidentified network models.

The authors of PQ-CNN presented a hardware-software framework for compressing and accelerating CNNs on FPGAs using product quantisation [64], adopting a similar idea to that used in Quantised CNN [149, 156]. Going further, the authors implemented an extra codebook to compress encoding parameters, increasing the compression of the original algorithm. During inference, since all possible multiplication outputs with every codeword are precomputed and stored on-chip, PQ-CNN sees dot products for both convolutions and fully connected layers converted into table lookups and accumulations. The authors' Amazon F1 implementation achieved 4.6kcl/s for the VGG16 model with a sub-0.5pp drop in top-five accuracy for ImageNet.

5.3 Low-rank Factorisation

5.3.1 Algorithmic Development. Post-training low-rank factorisation of DNNs can achieve significant network compression and computation reductions for inference. Denton et al. analysed the effect of applying several decomposition methods—singular-value decomposition (SVD), canonical polyadic (CP) decomposition, and biclustering approximation—on pretrained weight matrices [36]. A biclustering approximation performs k-means clustering on rows and columns of weight matrices [64]. These methods were tested with a 15-layer CNN classifying the ImageNet dataset. Among them, SVD achieved the best performance: 13× compression of the first fully connected layer with 0.84pp of top-one accuracy loss. Tai et al. also performed network decomposition using SVD [138]. They achieved up to 5.0× compression and a 1.8× throughput speedup for ImageNet on AlexNet, reporting a top-five accuracy reduction below 0.5pp.

While post-training decomposition is simple and flexible, many works have shown that training after decomposition can recover compression losses. As suggested by Jaderberg et al., weight matrices can be decomposed into several low-rank matrices to enable workload and/or memory reductions [63]. The authors proposed the factorisation of each of their four-dimensional layers into a sequence of two regular convolutional layers, each of three dimensions. Experiments with various nonstandard scene text character recognition datasets showed that this method achieved, on average, a 4.5× increase in throughput with around-1pp falls in accuracy for some unidentified networks. This factorisation scheme inspired MobileNet, which uses one three-dimensional "depthwise" and one two-dimensional "splitwise" separable convolutional layers to approximate each original layer [60]. Assume that a convolutional layer contains $K \times K \times M \times N$ values, where K, M, and N are the size of the kernel and numbers of input and output channels, respectively. In MobileNet, this is factorised into a depthwise convolutional layer with $K \times K \times M \times 1$ values and a pointwise convolutional layer of size $1 \times 1 \times M \times N$. This method effectively reduces the complexity of forward propagation from $O(MD^2K^2N)$ to $O(MD^2(K^2 + N))$, where D is the size of the input feature map. Experiments with ImageNet showed that MobileNet can achieve a 3.0pp top-one accuracy improvement with 46× compression for AlexNet.

Ba et al. combined low-rank factorisation with knowledge distillation, where a deep and complex neural network is mimicked with a simpler, shallower one [11]. More detail on knowledge distillation is given in Section 5.5. The authors noticed that learning is very slow for the weight matrices of shallow networks. Since there are many highly correlated parameters, gradient descent converges slowly, with the majority of training time spent on matrix-vector multiplication. They suggested that forward and backward propagation could be sped up by approximating each large weight matrix as the product of two low-rank matrices. Increases in convergence rate of the network mimicking and reductions in memory space complexity were observed. Lebedev et al. presented a CP decomposition-based retraining method facilitating greater workload reductions, achieving a 4.5× throughput boost with ∼1pp of top-five ImageNet accuracy loss for layer two of AlexNet [73].

Following the logic that learnt weight matrices tend to be structured and can be decomposed using low-rank factorisation, Denil et al. suggested the storage of only parts of weight matrices, predicting the remainder using a second learning model [35]. They reported that, in the best case—with small-scale datasets—more than 95% of weights can be predicted without accuracy loss. The networks used therein were nonstandard.

Rather than compressing layers individually, Kim et al. performed "one-shot" whole-network compression using Tucker decomposition. Here, the post-decomposision ranks of all layers are determined all at once through global Bayesian matrix factorisation. Experiments showed that, while this method requires at least 10 retraining epochs for accuracy recovery, the inference of AlexNet on an Nvidia Titan X GPU achieved 1.8× speedup, with 1.7pp of top-five ImageNet accuracy loss, against an FP32 baseline on the same platform.

5.3.2 Hardware Implementation. Low-rank factorisation methods produce structured DNN models that can be inferenced efficiently on GPPs with dense matrix-vector BLAS. Li et al. presented a CNN compression framework combining coarse-grained pruning using sparsification with low-rank factorisation [77]. Similar to the idea proposed by Jaderberg et al. [63], the authors represented filters as linear combinations of lower-rank basis filters. GPU experiments with AlexNet, GoogleNet, and VGGNet-A revealed about-2× throughput speedups without accuracy loss for ImageNet.

Custom hardware implementations, however, can achieve comparable performance with lower power envelopes. Rizakis et al. implemented SVD-factorised gates for LSTMs [115]. In their proposal, SVD is performed on the weights of the four LSTM gates independently. For each gate, the weights associated with both the current input and previous output are concatenated together to form a large weight matrix, which is then SVD-factorised. Pruning is also performed by retaining only rows with a majority of non-zeroes in each weight matrix. The authors implemented their design on an FPGA platform, achieving a 6.5× throughput increase for an arbitrarily chosen LSTM compared with an uncompressed FPGA-based LSTM baseline.

5.4 Structured Matrices

5.4.1 Algorithmic Development. A weight matrix can be represented as a structure of repeated patterns such that it can be expressed with fewer parameters. The use of circulant matrices for representing weight matrices $\boldsymbol{W}$ in CNNs and RNNs has proven to be a very popular proposal [26, 27, 93, 131, 146]. A circulant matrix $\boldsymbol{W}_{\text{circ}}$ of size $K \times K$ is square, with all rows being a shifted version of the first, $\boldsymbol{w}_{0*}$, thereby reducing spatial complexity from $O(K^2)$ to $O(K)$. It is constructed as such:

$$\boldsymbol{W}_{\text{circ}} = \begin{pmatrix} w_0 & w_{K-1} & \dots & w_2 & w_1 \\ w_1 & w_0 & w_{K-1} & & w_2 \\ \vdots & w_1 & w_0 & \ddots & \vdots \\ w_{K-2} & & \ddots & \ddots & w_{K-1} \\ w_{K-1} & w_{K-2} & \cdots & w_1 & w_0 \end{pmatrix}.$$

The multiplication of $\boldsymbol{W}_{\text{circ}}$ by input vector $\boldsymbol{x}$ can thus be computed using a fast Fourier transform (FFT) of the first row of $\boldsymbol{W}_{\text{circ}}$, reducing inference time complexity from $O(K^2)$ to $O(K \log K)$, as

$$\boldsymbol{W}_{\text{circ}}\boldsymbol{x} = \text{ifft}(\text{fft}(\boldsymbol{w}_{\text{circ}0*}) \odot \text{fft}(\boldsymbol{x})).$$

While the circulant matrix method has shown outstanding memory and computational complexity reductions, its application also introduces accuracy degradation. For example, the AlexNet implementation of a circulant matrix-based framework, CirCNN, achieved compression of 40× with

16-bit fixed-point quantisation, yet its use also resulted in 2.2pp of ImageNet accuracy degradation against an FP32 baseline [37]. An alternative transformation, the Adaptive Fastfood transform (AFT), achieved a compression ratio of 3.7, but only about 0.1pp of accuracy loss with ImageNet on AlexNet [154]. In an AFT, a weight matrix $\boldsymbol{W}$ is approximated as

$$\boldsymbol{W}_{\text{AFT}} = \boldsymbol{SHG\Pi HB},$$

in which $\boldsymbol{S}$, $\boldsymbol{G}$, and $\boldsymbol{B}$ are trainable diagonal matrices, $\boldsymbol{H}$ a Hadamard matrix, and $\boldsymbol{\Pi} \in \{0,1\}^{K\times K}$ a trainable permutation matrix. This and the circulant method have equal complexities.

For both of the aforementioned structures, generality is not guaranteed when dealing with classification tasks of varying scales. Sindhwani et al. proposed structured transformations characterised by the notion of a *displacement rank* parameter [131]. With different displacement ranks, a continuum is exposed from fully structured to completely unstructured. With displacement rank less than or equal to two, weight matrices become Toeplitz matrices, which have the form

$$\boldsymbol{W}_{\text{Top}} = \begin{pmatrix} w_0 & w_{-1} & \cdots & w_{-(K-2)} & w_{-(K-1)} \\ w_1 & w_0 & w_{-1} & & w_{-(K-2)} \\ \vdots & w_1 & w_0 & \ddots & \vdots \\ w_{K-2} & & \ddots & \ddots & w_{-1} \\ w_{K-1} & w_{K-2} & \cdots & w_1 & w_0 \end{pmatrix}.$$

Different to a circulant matrix, a Toeplitz matrix $\boldsymbol{W}_{\text{Top}}$ of size $K \times K$ has element values $w_{-(K-1)}$ to w_{K-1}. Matrix-vector multiplications can still take advantage of FFTs by embedding Toeplitz matrices into larger circulant matrices, as in

$$\boldsymbol{W}_{\text{circ, Top}} = \left(\begin{array}{ccccc|ccccc} w_0 & w_{-1} & \cdots & w_{-(K-2)} & w_{-(K-1)} & 0 & w_{K-1} & \cdots & w_2 & w_1 \\ w_1 & w_0 & w_{-1} & & w_{-(K-2)} & w_{-(K-1)} & 0 & w_{K-1} & & w_2 \\ \vdots & w_1 & w_0 & \ddots & \vdots & \vdots & w_{-(K-1)} & 0 & \ddots & \vdots \\ w_{K-2} & & \ddots & \ddots & w_{-1} & w_{-2} & & \ddots & \ddots & w_{K-1} \\ w_{K-1} & w_{K-2} & \cdots & w_1 & w_0 & w_{-1} & w_{-2} & \cdots & w_{-(K-1)} & 0 \\ \hline 0 & w_{K-1} & \cdots & \cdots & w_1 & w_0 & w_{-1} & \cdots & \cdots & w_{-(K-1)} \\ w_{-(K-1)} & 0 & w_{K-1} & & w_2 & w_1 & w_0 & w_{-1} & & \vdots \\ \vdots & w_{-(K-1)} & 0 & \ddots & \vdots & w_2 & w_1 & w_0 & \ddots & \vdots \\ w_{-2} & & \ddots & \ddots & w_{K-1} & \vdots & & \ddots & \ddots & w_{-1} \\ w_{-1} & w_{-2} & \cdots & \cdots & 0 & w_{K-1} & \cdots & \cdots & w_1 & w_0 \end{array}\right),$$

and exploiting the relationship

$$\boldsymbol{W}_{\text{Top}}\boldsymbol{x} = \begin{pmatrix} \boldsymbol{I}_K & \boldsymbol{0}_{K\times K} \end{pmatrix} \boldsymbol{W}_{\text{circ, Top}} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{0}_{K\times K} \end{pmatrix},$$

wherein $\boldsymbol{I}$ and $\boldsymbol{0}$ are identity and zero matrices, respectively [45].

A family of Toeplitz-like matrices can be generated by increasing rank beyond two. With rank K, a matrix becomes unstructured and uncompressed. Lu et al. applied Toeplitz-like matrices in LSTMs, with weight matrices of gates trained in Toeplitz-like structures of various ranks [93]. The authors compressed the first two layers of an unidentified five-layer LSTM into structures of rank five, achieving a compression ratio of around 1.7 with a ~0.3pp loss in speech recognition accuracy for a dataset consisting of some 300h of English utterances.

While the authors of the works mentioned above reported that the use of circulant matrix-based methods resulted in the incursion of at-least 2pp accuracy drops for large-scale CNN image classifications, their accuracies for RNN tasks are significantly superior. Wang et al. implemented circulant matrices together with non-linear function approximation and quantisation for LSTMs [146]. Language modelling and speech recognition were performed by their 90nm ASIC, achieving more than 20× compression with a 2.8pp loss in accuracy for classification of the AN4 speech database.

C-LSTM features block-circulant matrices, each of which consists of circulant submatrices of arbitrary size [145]. Tunable block size facilitates a tradeoff between storage requirements and accuracy. Experiments with the Google LSTM architecture revealed a linear relationship between block size and compression ratio, as well as a clear tradeoff between block size and TIMIT phone error rate (PER) increase. For an LSTM model with block size of eight on the TIMIT dataset, C-LSTM exhibited 7.6× compression and a 2.6× workload reduction while incurring a 0.32pp PER rise.

5.4.2 Hardware Implementation. Convolutions on GPPs are normally performed after unrolling, flattening four-dimensional inputs and kernels into two-dimensional matrices. This converts four-dimensional tensor operations into two-dimensional matrix multiplications, trading off memory use for performance. For block-circulant matrix methods, since each two-dimensional slice of a kernel is circulant, the two-dimensional unrolled version of that kernel is also block-circulant. Time complexity reductions from the FFT-based method for block-circulant matrix inference are hence achievable for DNN inference performed on both GPPs and in custom hardware. Despite this, custom hardware implementations still excel in terms of energy efficiency [37].

Combined with 16-bit fixed-point quantisation, FPGA-based C-LSTM achieved a 10× throughput speedup and 34× energy efficiency improvement over ESE for the Google LSTM, the prior state of the art [145]. Ding et al. presented implementations using similar methods for CNNs and RNNs on both FPGAs and ASICs [38]. Using Intel Cyclone V FPGAs, the authors achieved at-least 150× and 72× improvements in performance and energy efficiency, respectively, over IBM TrueNorth implementations [2] of some unidentified networks. For Xilinx Kintex UltraScale FPGA LSTM implementation, the proposed architecture achieved up-to 21× and 34× improvements in throughput and energy efficiency, respectively, over ESE for the Google LSTM [52]. The authors also experimented with a LeNet-5 ASIC implementation, achieving a throughput of 1.1Mcl/s and energy efficiency of 8.1Mcl/J. Wang et al. presented a circulant matrix-based LSTM inference implementation in 90nm ASIC technology [146]. They adopted a hybrid strategy in their work, also exploiting fixed-point quantisation and activation function approximation. With a 520KiB on-chip memory allocation, the authors were able to process a 512×512 compressed layer of an arbitrarily chosen LSTM in 1.7µs: equivalent to 580kcl/s.

Fox et al. implemented AFTs for accelerating matrix-vector multiplication on FPGAs [41]. Although their work was not presented in the context of DNN inference, its results on matrix-vector multiplication are still relevant. The authors concluded that the AFT's small memory complexity allows for the processing of input matrices some 1,000× larger than previous online kernel methods with the same area occupancy.

5.5 Knowledge Distillation

5.5.1 Algorithmic Development. Knowledge distillation mimics large, complex DNNs using simpler and shallower networks to achieve network compression. In one of the earliest works in this field, Hinton et al. suggested that knowledge could be distilled from an ensemble of models (teachers) into a simple model (student) by training the student model with outputs from the teachers [59]. Ba et al. provided empirical evidence showing that, in simple machine-learning tasks, a

student network can mimic teacher networks with comparable performance [11]. In FITNet, intermediate outputs of these teacher models are used as "hints" for training the student model to improve its accuracy [116]. Experiments with the CIFAR-10 dataset showed that a FITNet trained from an unidentified 9M-parameter teacher CNN could achieve 10× compression and a 1.4pp accuracy improvement vs. the teacher network. The authors explained that a reduction in network complexity from teacher to student led to less overfitting, causing the accuracy increase.

Chen et al. proposed various optimisations for improving the performance of network mimicking [20]. Unlike carefully selected image classification datasets with uniform class distributions, object detection problems need to deal with dominant background classes. Class-weighted cross-entropy can be introduced to handle such scenarios, wherein a background class is assigned an appropriate scaling factor to correct for class imbalances. When teacher overfitting occurs, hints from a teacher network may "mislead" a student into even more severe overfitting. In an effort to avoid this, Chen et al. used their teacher network's original regression curve as an upper bound for student network training. Experiments with the PASCAL, KITTI, and COCO datasets showed that these optimisations improved accuracies by 3–5pp.

Alemdar et al. introduced a framework for knowledge distillation in which ternary student networks were trained from a ternarised teacher [5]. During ternarisation, two thresholds for each weight index i, $\{\delta_i^-, \delta_i^+\}$, are used to differentiate quantisation levels $\{w_i^-, w_i^+\}$ from zero. The authors suggested that the use of well selected thresholds should result in outputs from the student network perfectly matching those of the teacher network. A greedy method was proposed to search for thresholds by minimising the difference between the probability distribution functions of layer-wise outputs from the student and teacher networks. Experiments with MNIST, CIFAR-10, and SVHN showed that this work achieved higher accuracies than IBM TrueNorth classifying the same datasets on VGG-like models with ternary data [2].

5.5.2 Hardware Implementation. Knowledge distillation essentially converts deep DNNs into shallow ones, which, from a hardware perspective, allows the replacement of deep, sequential processing with parallel, distributed processing. This structural conversion greatly facilitates the acceleration of DNN training and inference using GPPs. Ba et al. even observed that some shallow, mimicked models reached similar accuracies for TIMIT to deep models about 8× more quickly [11].

While some acceleration can be achieved with knowledge distillation on GPPs, further benefit can be realised given the flexibility of custom hardware by taking advantage of additional approximation. Alemdar et al. presented a hardware mapping framework in which student networks trained through network mimicking are translated into hardware descriptions for FPGA or ASIC implementation [5]. Their Xilinx Virtex-7 FPGA prototype achieved an over-30× throughput improvement and comparable energy efficiency vs. IBM TrueNorth [2] executing VGG-like models. A 28nm ASIC implementation was also presented and compared against a state-of-the-art ASIC implementation, EIE [53]. While their ASIC did not beat EIE in terms of throughput, it did achieve 1.2× energy efficiency and 2.9× area occupancy improvements for an unidentified network model.

6 INPUT-DEPENDENT COMPUTATION

6.1 Algorithmic Development

Different regions of a DNN's input data may have differing levels of contribution to its output. Input-dependent computation exploits this observation by assigning compute proportionally to the input data's relative importance. Stochastic Times Smooth units mask CNN input frames with a pretrained binary decision matrix to facilitate conditional computation, which was shown to give 10× compression for a nonstandard CNN classifying the MNIST dataset with a 0.2pp accuracy improvement [14]. Karpathy et al. allocated more resources to the centres of CNN input frames for

improved video classification accuracy [68]. Their implementation consists of two CNNs in parallel, with a "context stream" CNN processing entire frames and a "fovea stream" CNN processing only the centre of each. The authors reported a 65% prediction accuracy on the UCF-101 video prediction dataset: state-of-the-art performance at the time of publication.

Low-rank approximation has not just been studied in the parameter space; it has been used for input compression as well. In Deep3 [118] and DeLight [117], input data matrices are factorised into lower-rank matrices using an "embedding matrix." These are iteratively updated to reduce the Frobenius norm of factorisation errors. Experiments with Deep3 on GPUs on various deep-learning tasks, including audio classification, demonstrated up-to 11× inference speedups compared to a TensorFlow baseline running the same models [118].

While the aforementioned *static* computation allocation schemes can achieve significant resource savings and/or accuracy improvements, recent research, such as Dynamic Capacity Networks, has introduced *dynamic* input-dependent allocation, guided at runtime by additional pretrained subnetworks [6]. In Bengio et al. [13] and Liu et al.'s [89] proposals, and Runtime Neural Pruning (RNP) [85], partial execution of DNNs is performed using pretrained Markov decision process reinforcement learning. RNP was shown to achieve a 10× workload reduction and 5.9× latency reduction for VGG16 with ImageNet in return for a 4.9pp drop in top-five accuracy. Runtime methods achieve superior accuracy to their static counterparts at the expense of an extra network.

The works discussed above all targetted CNNs, for which computation is dependent upon the spatial features of their inputs. The authors of DeltaRNN, however, reduced RNN workload based on inputs' temporal behaviour [43]. DeltaRNN updates the output of an RNN only when its input changes by more than some threshold. They reported 9.8× throughput and 130× efficiency improvements for an arbitrarily chosen network, with a 1.5pp accuracy drop, against their baseline classifying the TIDIGITs dataset.

6.2 Hardware Implementation

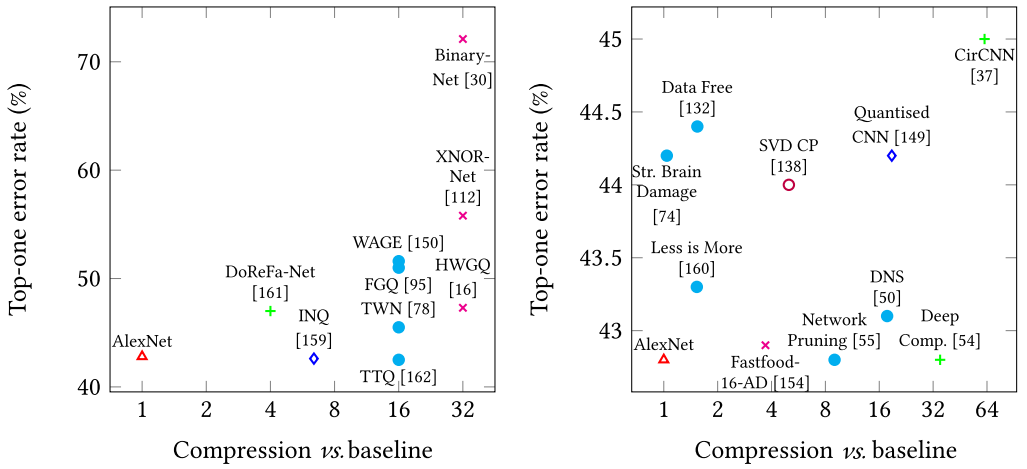
Since input-dependent computation involves frequent dynamic branching during inference, these implementations are not likely to pipeline efficiently, especially for deep CNNs. Hence, for CNN implementations exploiting this method, throughput is not their greatest advantage. They are instead focussed more on latency-critical applications, which generally do not require high throughput. Custom hardware, unlike GPPs, allows for specially designed dynamic branching mechanisms that can inference fine-grained, irregular data patterns more efficiently.

The authors of CascasdeCNN presented the input-dependent computation of CNN inference on FPGAs [71]. Similar to Dynamic Capacity Networks [6], CascadeCNN features a high-precision subnetwork in addition to a low-precision main network. The former is activated when there is a potential misclassification in the latter, i.e., when the confidence of the main network's best guess is low. Experiments showed that CascadeCNN achieved latency reductions of up to 55% for VGG-16 and 48% for AlexNet over the baseline design for the same resource budget and accuracy with ImageNet. The FPGA implementation of DeltaRNN on an LSTM requiring 5.6Gop/cl demonstrated reduced off-chip memory bandwidth, achieving a throughput of 220cl/s and an energy efficiency of 29cl/J: state-of-the-art performance for RNN inference at the time [43].

7 ACTIVATION FUNCTION APPROXIMATION

7.1 Algorithmic Development

With non-linear functions such as sigmoid and tanh, computations including exponentiation and division are expensive to perform. Piecewise Linear Approximation of Non-linear Functions (PLAN) simplifies such functions into serieses of table lookups [8]. In turn, this leads to the



(a) Quantisation methods: baseline (△), eight-bit fixed point (+), logarithmic (◊), ternary (●), and binary (×).

(b) Weight-reduction methods: baseline (△), hybrid (+), weight sharing (◊), pruning (●), structured matrix (×), and factorisation (○).

Fig. 3. Comparison of reported top-one error rates for implementations of AlexNet classifying ImageNet.

quantisation of activations in subsequent layers, reducing both memory requirements and numbers of arithmetic operations to perform. PLAN appears more often in RNN implementations than CNNs; mainstream CNNs use ReLU as the activation function, which can be cheaply implemented by comparing outputs with zero. In RNNs, however, empirical analysis suggests that sigmoid and tanh provide better performance, whereas ReLU not only performs poorly but also diverges frequently, partly because it is positively unbounded [39].

7.2 Hardware Implementation

PLAN can be efficiently implemented in custom hardware. Guan et al. implemented PLAN within an FPGA-based inference framework for unidentified LSTMs, and their experiments showed that its use introduced only 0.63pp of TIMIT accuracy degradation [46]. Li et al. [82] and the authors of ESE [52], C-LSTM [145], and DeltaRNN [43] implemented arbitrarily chosen RNNs on FPGAs with PLAN, reporting increases in throughput with negligible accuracy losses for the same dataset.

8 TRADEOFFS AND CURRENT TRENDS

Thus far, we have detailed DNN approximation techniques and their hardware implementations on different platforms. Performance evaluations were made against benchmarks and baseline implementations of their authors' choosing, which are inconsistent and often not particularly useful when attempting to perform comparisons. We now quantitatively evaluate the hardware and software performance of those works using common DNN models and datasets as benchmarks. By doing so, we analyse the compression-accuracy tradeoffs of the approximation techniques and their design-space exploration for custom hardware, from which we explain current research trends.

8.1 Compression vs. Accuracy

Figure 3(a) compares the compression-accuracy behaviour of key quantisation methods introduced in Section 4 for ImageNet on AlexNet, indicating a clear relationship between precision and error

rate. Among the methods, binary networks exhibit greater accuracy degradations (≥4.5pp) than the remainder (<3.0pp), while also achieving the greatest compression ratios: 32 vs. an FP32 baseline.

The parameters of trained DNNs usually have Gaussian-like distributions, wherein the majority of data have near-zero values. For this reason, binary networks exhibit high quantisation error for values with small magnitudes because they are unable to represent zeroes. Compared to binarisation, ternarisation generally results in better accuracy, with compression ratios of 16. Among all methods compared, TTQ has the highest accuracy at a reasonably high compression ratio, suggesting that the ability to represent zeroes has significant implications for network performance [162]. INQ reached a similar level of accuracy to TTQ, but with a lower compression ratio (6.4) [159]. The accuracy of INQ is higher than fixed-point-quantised networks with similar precisions, supporting the conclusion by Lai et al. that it is weights' representation range, rather than precision, that is crucial to the preservation of accuracy [72].

Figure 3(b) facilitates comparison of the compression-accuracy tradeoffs, also for ImageNet on AlexNet, of the key weight-reduction methods introduced in Section 5. It shows that the reported compression ratios for weight-sharing methods, such as Deep Compression [54] and Quantised CNN [149], and structured matrices, e.g., CirCNN [37], are higher than the alternatives. This observation supports the theoretical analysis in Sections 5.2 and 5.4 that these methods have good memory complexity reduction capabilities.

Structured matrix methods induce significant accuracy degradation in CNNs [37] but not so much in LSTMs [37, 145]. This phenomenon is not yet well understood.

Pruning-based methods also lead to the obtainment of good accuracies at high compression ratios. Among them, fine-grained methods (DNS [50] and Network Pruning [55]) show more promising tradeoffs than coarse-grained alternatives (Structured Brain Damage [74] and Less is More [160]). This suggests that higher pruning granularities, despite inducing significant irregularity, possess greater potential for network compression and memory transfer reductions.

Deep Compression exhibited both outstanding accuracy and compression [54]. As a hybrid strategy, multiple quantisation methods work together to provide high compression.

We can conclude that (re)training has proven to be effective in compensating for accuracy losses incurred due to approximation [55, 122]. The authors of methods exploiting binarisation, ternarisation, structured matrices, low-rank factorisation, and knowledge distillation trained their networks from scratch, while the remaining methods—apart from Data Free [132]—use post-approximation retraining. Although Data Free featured pruning of similar neurons without the employment of retraining, it was used for all of the implementations in Figure 3, suggesting that retraining has become a standard accuracy-recovery approach in state-of-the-art proposals.

8.2 Design-space Exploration

Table 1 shows how each approximation method contributes to DNN inference acceleration in custom hardware. Increases in parallelism and reductions in model memory use increase compute bounds and arithmetic intensities, respectively, which, in turn, increase throughput.

Quantisation-based methods allow for increased parallelism through the use of cheaper arithmetic units. They also facilitate memory transfer reductions. With extremely low-precision quantisation, it becomes feasible to fix parameters in hardware such that weights do not need to be stored in, or fetched from, off-chip memory. Weight-reduction methods reduce numbers of parameters, saving memory while simultaneously decreasing workload. Weight sharing is slightly different from the other weight-reduction methods, because it does not necessarily cause a reduction in workload. The number of operations to be performed per classification can be reduced if results are precomputed and stored on-chip, such as in PQ-CNN, however [156]. Unlike weight-reduction methods, input-dependent methods reduce workload without decreasing memory

Table 1. How Each Approximation Method Contributes to DNN Inference Acceleration in Custom Hardware

		Cheaper arithmetic operations	Memory reduction	Workload reduction
Quantisation	Fixed-point representation	✔	✔	✘
	Binarisation and ternarisation	✔	✔	✘
	Logarithmic quantisation	✔	✔	✔ (if shift lengths are constant)
Weight reduction	Pruning	✘	✔	✔
	Weight sharing	✘	✔	✔ (if multiplications are precomputed)
	Low-rank factorisation	✘	✔	✔
	Structured matrices	✘	✔	✔
	Knowledge distillation	✘	✔	✔
	Input-dependent computation	✘	✘	✔
	Activation function approximation	✘	✘	✔
	Hybrid strategies	✔	✔	✔

occupancy. Through precomputation, activation function approximation only reduces workload. Hybrid strategies have been commonly adopted recently; these can benefit from all three factors, achieving greater performance than could be realised through the use of any single method.

8.2.1 Throughput. Table 2 details the performance of state-of-the-art FPGA-based DNN inference engines targetting the CIFAR-10 (CNN), ImageNet (CNN), and TIMIT (RNN) datasets. Implementations are ordered according to power consumption, thus platforms of similar scales are adjacent. While categorised with respect to their target datasets, frameworks accelerating the inference of the same dataset may have been benchmarked using different DNN models and hence with dissimilar workloads. Some works did not report full-network workload information, making it impossible for us to quantify their throughputs. We thus detail arithmetic performance, which captures raw computational speed, as well.

In general, custom hardware implementations exhibit up-to orders-of-magnitude higher throughput than GPP equivalents of similar scales, corresponding to the conclusions drawn in Section 3. Among the custom hardware implementations, the throughput of ASIC platforms is higher than other works with similar power consumption, largely due to their higher clock frequencies.

By comparing Wang et al. [144] and Zhao et al.'s [158] CIFAR-10-targetting CNN implementations with the Going Deeper [111], fpgaConvNet [142], and FP-BNN [83] ImageNet CNNs, all of which used FPGAs of similar scales, we can observe that, as precision is reduced, linear or even superlinear throughput increases can be achieved. Superlinear increases can be explained using the roofline modelling in Section 3. With quantisation on FPGAs, the use of cheaper fixed-point processing units allows for increased parallel-computing capability via area savings, in turn leading to increases in compute bounds. Arithmetic intensity can also be increased as model size decreases due to the opportunities presented by on-chip caching. The combined effect of these factors allows inference throughput to increase linearly if the baseline is memory bound, or superlinearly

Table 2. Comparison of Large-scale DNN Inference Performance

		Quantisation(s)[1]		Platform	Frequency (MHz)	Throughput (cl/s)	Workload (Gop/cl)	Arithmetic perf. (Gop/s)	Efficiency (cl/J)	Approximation method(s)[1]
		Weights	Acts							
CNN (CIFAR-10)	Wang et al. [144]	FXP8	FXP8	Xilinx Zynq XC7Z020	100	103	0.0248	2.56	54.4	FXP
	Zhao et al. [158]	BIN	BIN	Xilinx Zynq XC7Z020	143	168	1.24	208	35.6	BIN
	CaffePresso [58]	FXP32	FXP32	Adapteva Parallella	–	95.9	0.0146	1.40	14.2	–
	FINN [141]	BIN	BIN	Xilinx Zynq XC7Z045	200	21900	0.113	2500	3160	BIN
	CaffePresso [58]	FXP16	FXP16	TI Keystone-II	–	1000	0.0146	146	1000	FXP
	FP-BNN [83]	BIN	BIN	Intel Stratix V 5SGSD8	150	7640	1.23	9400	292	BIN
	CPU [83]	FP32	FP32	Intel Xeon E5-2640	2500	147	1.23	181	1.55	–
	GPU [83]	FP32	FP32	Nvidia Tesla K40	745	1510	1.23	1850	6.41	–
CNN (ImageNet)	YodaNN (0.60V) [9]	BIN	FXP12	65 nm ASIC	–	4.50	3.60	16.2	13400	BIN
	DaDianNao [22][2]	FXP16	FXP16	28 nm ASIC	606	–	–	452	–	FXP
	EIE [53][2]	FXP4	FXP16	45 nm ASIC	800	–	–	3000	–	FXP, PRU, WS
	NeuFlow [40][2]	FXP16	FXP16	45 nm ASIC	400	–	–	160	–	FXP
	fpgaConvNet [142]	FXP16	FXP16	Xilinx Zynq XC7Z045	125	5.07	30.7	156	0.726	FXP
	Angel-eye [48]	BFP8	BFP8	Xilinx Zynq XC7Z045	150	6.12	30.7	188	0.635	BFP
	Going Deeper [111]	FXP16	FXP16	Xilinx Zynq XC7Z045	150	4.46	30.7	137	0.463	FXP
	Li et al. [81]	–	–	Xilinx Zynq XC7Z045	–	205	1.33	272	–	PRU
	Shen et al. [127]	FXP16	FXP16	Xilinx Virtex US VCU440	200	26.7	30.7	821	1.03	FXP
	FP-BNN [83]	BIN	BIN	Intel Stratix V 5SGSD8	150	863	2.27	1960	33.0	BIN
	TPU [66][2]	FXP8	FXP8	28 nm ASIC	700	–	–	92000	–	FXP
	HARPv2 [100]	BIN	BIN	Intel HARPv2	–	114	30.7	3500	2.37	BIN
	GPU [100]	FP32	FP32	Nvidia Titan X	–	121	30.7	3710	1.76	–
	Brainwave [28]	BFP5	BFP5	Intel Arria 10	300	559	7.80	4360	4.47	BFP

(Continued)

Table 2. Continued

		Quantisation(s)[1]		Platform	Frequency (MHz)	Throughput (cl/s)	Workload (Gop/cl)	Arithmetic perf. (Gop/s)	Efficiency (cl/J)	Approximation method(s)[1]
		Weights	Acts							
RNN (TIDIGITs)	Wang et al. [146]	LOG8	FXP8	90 nm ASIC	600	585000	0.00421	2460	580000	FXP, LOG, ACT, STR
	DeltaRNN [43]	FXP16	FXP16	Xilinx Zynq XC7Z100	125	2650000	0.000453	1200	362000	FXP, ACT, IDC
	C-LSTM FFT8 [145]	FXP16	FXP16	Xilinx Kintex US XCKU060	200	195000	0.208	40600	8130	FXP, ACT, STR
	ESE [52]	FXP12	FXP16	Xilinx Kintex US XCKU060	200	12100	0.208	2520	296	FXP, ACT, PRU
	CPU [52]	FP32	FP32	Intel i7-5930K	–	166	0.208	34.6	1.50	–
	Brainwave [28]	BFP5	BFP5	Intel Stratix 10	250	13500	1.67	22600	108	BFP
	GPU [52]	FP32	FP32	Nvidia Titan X	–	4160	0.208	866	20.6	–

Implementations are ordered by power consumption, lowest first.
[1]FXP: fixed point. BFP: block floating point. BIN: binary. LOG: logarithmic. ACT: activation function approximation. PRU: pruning. STR: structured matrix. WS: weight sharing. IDC: input-dependent computation.
[2]Reported arithmetic performance is a "peak" value, not that for any particular network, since the authors did not report the latter.

if compute bound. The accuracy-throughput tradeoff exposed through quantisation makes it possible for embedded-scale custom hardware implementations to beat even high-end GPPs in terms of inference throughput. This is evident throughout Table 2, in which the performance of schemes employing binarisation on custom hardware can be seen to have achieved either superior or comparable throughput to that of popular high-performance GPPs.

EIE [53] and Li et al.'s work [81] used pruning with fixed-point quantisation in ASICs and FPGAs, respectively, for CNN weight reduction. Comparing these against other works listed that used the same platform but without pruning, NeuFlow in ASICs [40] and Going Deeper in FPGAs [111], significantly superior arithmetic performance was obtained. This supports the other conclusion drawn from the roofline modelling in Section 3: with network compression, operational intensity increases due to reduced off-chip memory traffic, facilitating speedups. EIE, using fine-grained pruning with runtime zero-skipping, achieved a 19× improvement in arithmetic performance over NeuFlow, whereas Li et al.'s work, using coarse-grained pruning, achieved only 2× improvement over Going Deeper. This supports the conclusion in Sections 3 and 8.1 that fine-grained pruning results in more workload reduction than coarse-grained, and that custom hardware allows for the design of efficient mechanisms to convert these reductions into speedups.

As mentioned in Section 5.4, circulant matrix-based methods do not work well with CNNs due to their significant accuracy losses, yet they provide exceptionally good accuracy and compression for RNNs. This is reflected in Table 2, in which it is shown that C-LSTM exhibited 47× and 390× gains in throughput and efficiency, respectively, compared to a GPU implementation [145]. Among all RNN implementations listed, those that employed block-circulant matrices or input-dependent computation achieved superior throughputs and efficiencies *vs* the remainder since the use of these methods resulted in the greatest workload reductions.

Almost all of the listed RNN FPGA frameworks made use of hybrid strategies, featuring processing elements tailored to low-precision computation along with weight reduction, achieving significant throughput improvements compared to GPU alternatives.

8.2.2 Latency. While the majority of existing works in the field are throughput- or energy-oriented, some DNN applications prioritise latency instead. Some implementations simultaneously achieved good throughput and latency performance. Ma et al. implemented VGG-16 on FPGAs with fixed-point quantisation for ImageNet classification [94]. Tradeoffs between resource consumption and throughput were systematically analysed, with high performance achieved by balancing memory traffic and computation. The authors reported throughput of 21cl/s and latency of 48ms, both of which are 4.7× higher than the previous state of the art, Going Deeper [111].

The earliest version of fpgaConvNet was throughput-oriented [142]. The authors later extended their design-space exploration tool to optimise for latency in addition to throughput, demonstrating outstanding latency-critical application performance vs. alternative embedded implementations [143]. Zhang et al. also presented an FPGA-based RNN/CNN inference framework, providing highly configurable layer templates and a design-space exploration engine for resource allocation management facilitating design optimisation for resource-constrained latency minimisation [157].

Hardware implementations of input-dependent computation methods have an intrinsic emphasis on latency. Due to their conditional computation nature, pipeline stalls happen frequently, reducing throughput. This is not a problem for latency-driven applications, however, in which the inference batch size is normally one. Implementations based on input-dependent methods, e.g., CascadeCNN [71], are able to achieve significant latency reductions.

8.2.3 Energy Efficiency. Table 2 also facilitates the energy efficiency comparison of DNN inference implementations. Given a constant power budget, higher throughput translates to higher energy efficiency. Thus, approximation methods leading to higher parallelism and workload and/or

off-chip memory transfer reductions, such as binarisation [9], logarithmic quantisation [146], and block-circulant matrices [145], tend to result in higher energy efficiencies over alternative techniques with comparable network topologies and power consumptions.

When comparing platforms with similar throughput, the efficiency of power-hungry high-end GPPs tends to be lower than custom hardware implementations'. These facilitate parallelism at low precisions, achieving high throughput when running at a few hundred MHz, while CPUs and GPUs tend to operate at speeds on the order of GHz. For example, a binary HARPv2 implementation can provide comparable throughput to a Titan X Pascal GPU's, but is 24% more energy efficient [100].

The ASIC implementations achieve the highest energy efficiencies, primarily because they are not configurable and thus have lower capacitive loading than FPGA equivalents. Due to hardware overheads allowing for arbitrary logic and routing configurations and their lack of clock tree customisability, FPGAs can never compete with ASICs in terms of energy efficiency, yet FPGA implementations are still significantly more efficient than GPPs [7]. Memory hierarchy customisability also facilitates efficiency improvements, as was shown for YodaNN [9].

8.3 Application-specific Considerations

8.3.1 Retraining Time and Parameter Fine-tuning. Fixed-point and logarithmic quantisation, pruning, and input-dependent compute methods require post-approximation retraining. The majority of the pruning methods captured in Figure 3(b) use l_1 and l_2 regularisers. Their employment, however, tends to result in more iterations being required to achieve convergence, increasing training time. Ullrich et al. reported that training of networks exploiting the soft weight-sharing method is very slow for large-scale datasets [140]. Furthermore, the search for so-called *hyper-parameters*, such as pruning thresholds and quantisation precisions, can be cumbersome and expensive [55, 69].

The use of low-rank factorisation tends to necessitate more retraining iterations for convergence than alternative methods, since layer-wise factorisation results in increased network depth, exacerbating the problem of vanishing gradients in DNNs. Factorisation is also compute-intensive.

8.3.2 Parameterisation. During hardware design-space exploration, ASIC designs and some early FPGA-based works were only optimised for a single design metric: usually throughput. Many recent FPGA-based works have introduced general-purpose DNN accelerator frameworks that can cater to different design considerations based on desired application requirements. As a follow-up to FPGA-based framework fpgaConvNet [142], Stylianos et al. extended their automatic design-space exploration algorithm to also support area and latency optimisation [143].

8.3.3 Hardware Design and Turnaround. Due to the rapidly evolving landscape of DNN algorithmic development, the flexibility of the hardware design process has become a practical issue. With a time- and resource-consuming process, an inference platform could well become obsolete before it is manufactured. The design, fabrication, and validation of ASICs normally take months, if not years, to complete. Such slow turnarounds expose DNN application designers to high risks in terms of time and monetary investment. GPPs, however, are well supported by full-stack DNN design frameworks using high-level front ends, with which approximation methods can be prototyped in weeks. Compared with these two families of platforms, FPGAs provide a useful tradeoff between performance and design costs. High-level synthesis tools reduce design difficulty and lead time while allowing the obtainment of high throughput and energy efficiency.

8.3.4 Regularisation. The authors of works exploiting many approximation methods, including low-precision quantisation [31, 101, 108], pruning [55, 122], and weight sharing [21], reported accuracies greater than FP32 baselines after their application. Courbariaux et al. explained that

low-precision quantisation limits network capacity, forcing networks to leave local minima and find broader minima instead, improving generalisability by avoiding overfitting [31]. Similarly, in FITNet, the student network achieved 10× compression but a 1.4pp accuracy improvement over its teacher due to the regularisation effect from reduced network complexity [116]. The authors of HashedNets explained that the random "virtual" connections generated by their parameter hashing increased network expressiveness [21]. Similar to dropout layers in DNN training, the introduction of randomness from approximation, in the form of either quantisation noise or connections, creates regularisation that improves the accuracy of smaller networks.

9 FUTURE DIRECTIONS

Now that we have evaluated the current trends in the field of DNN approximation algorithms and their implementations, we are in a position to propose some promising future research directions.

9.1 Evaluation Methodologies

In the development of throughput-oriented DNN algorithm implementations, being able to identify bottlenecks is crucial to the efficiency of research. A misidentification of a bottleneck's source usually leads to wasted design effort. In many publications to date, authors have employed *ad hoc* evaluation methodologies, reporting improvements against seemingly arbitrary DNN benchmarks without systematically determining their baselines' bottlenecks, how the characteristics of the selected models affect those bottlenecks, or how far away design points are from theoretical maxima.

One of the major issues with DNN evaluation is the emphasis currently placed by many authors on peak arithmetic performance (in op/s). For example, the authors of the TPU stated that their architecture can achieve 92Top/s [66]. When tested with real DNN layers, however, that actually achieved was below 15Top/s due to memory bandwidth limits for all cases but one with a particularly high operational intensity. A focus on peak op/s can potentially lead to ignorance of the importance of microarchitectural design, making post-deployment accelerator efficiency underwhelming.

In Section 3, we compared the acceleration potential of DNN inference platforms using roofline modelling. For cross-platform evaluation, such models are useful since they present major bottlenecks in uniform and comparable formats, allowing the relative strengths and weaknesses of those platforms to be contrasted. Some authors have extended roofline modelling to capture other metrics. For example, in an attempt to analyse the tradeoff between energy efficiency and performance, Sayed et al. added frequency as a third axis, allowing power draw estimation [10].

For comparison of *implementations*, however—particularly those on the same platform—we are of the opinion that the use of roofline modelling is misguided. While points showing achieved arithmetic performance could be added to roofline plots, showing how much of their compute and memory bandwidth potential particular implementations achieve, the methodology's inherent orientation to arithmetic performance obscures other factors affecting analysis: chiefly workload. Two otherwise identical implementations with different levels of pruning, for example, may well exhibit negatively correlated op/s and cl/s, potentially making comparison of arithmetic performance misleading. In an attempt to tackle this, metrics including "equivalent throughput" (the arithmetic performance of a post-pruned network using the pre-pruning workload) have been introduced and are unfortunately now commonplace [37, 53]. We consider these to be unmeaningful and to needlessly distract from consideration of fundamental measures, particularly classification rate.

We encourage the community to report sustained throughput (in cl/s or similar) for standard, up-to-date models and datasets in preference to (peak) arithmetic performance. In conducting the

research for this article, we encountered many issues with performance comparison owing to authors evaluating their works very differently, with some of the benchmarks used unpopular or even obsolete. Emerging benchmark suites such as MLPerf and DeepBench, which provide selections of widely accepted and current test cases, should be used for comprehensive evaluation, thereby also facilitating apples-to-apples comparison.

9.2 Research Objectives

9.2.1 Convergence Guarantees and Optimal Design Choices. Many approximation methods do not yet have mathematical proofs of guaranteed convergence, meaning that existing methods may not be applicable to new DNN models. We are therefore of the opinion that theoretical investigation into each such method's convergence would be a very useful endeavour. As a counterexample, Li et al. provided derivations for quantised DNNs' convergence criteria [79]. Sakr et al. also investigated analytical guarantees on the numerical precision of DNNs with fixed-point quantisation [119].

It would also be interesting to prove the existence of optimal design choices for each method. For example, Tai et al. [138] suggested that the CP decomposition proposed by Lebedev et al. [73] does not guarantee an optimal rank-r factorisation since the problem of finding the best low-rank CP factorisation is ill-posed [33]. Similarly, for circulant matrix methods, we can clearly observe a difference in accuracy degradation between CNNs and RNNs, but it is not yet possible to explain this discrepancy mathematically. A good understanding of the convergence and applicability of the various approximation methods would be beneficial to allow for their generalisation.

9.2.2 Self-adaptive Hyper-parameter Fine-tuning. During quantisation and pruning, many hyper-parameters need to be determined through extensive manual fine-tuning with a validation dataset. This will become infeasible as networks deepen. Those with dynamic fine-tuning mechanisms are therefore potentially more scalable than those requiring manual intervention. As examples of the former, Bengio et al. [13] and Lin et al. [85] made pruning decisions using a Markov decision process, Liu et al. performed filter pruning using trainable scaling factors [92], Shin et al. learnt quantisation granularities via retraining [128], and Yang et al. removed filters to meet resource constraints [153]. If self-adaptive network fine tuning can be generalised to different hyper-parameters and network models, the latency of DNN application design could be significantly reduced.

9.2.3 FPGA-ASIC Heterogeneous Systems. From Table 2, we can conclude that, while FPGAs are extremely flexible, ASICs offer the greatest performance. Instead of focussing on purely FPGA- or ASIC-only solutions, Nurvitadhi et al. proposed the single-package, heterogeneous integration of FPGAs and ASICs using Intel's Embedded Multi-die Interconnect Bridge [103]. In their system, the ASIC components, called TensorTiles, execute typical DNN operations such as matrix-vector MACs at eight-bit or lower precision, while the FPGA enables the application-specific optimisation of data management and scheduling. With two TensorTiles and one FPGA, this design demonstrated 3.3× and 4.0× improvements in energy efficiency and throughput, respectively, with AlexNet against an FPGA-only implementation on an Intel Stratix 10. This work proved that such heterogeneous systems are promising platforms for DNN applications and thus deserve particular attention. Xilinx's recently announced Adaptive Compute Acceleration Platform, featuring a hardened array of processors suited to neural network compute interfaced with soft logic through a network on chip, was designed to simultaneously achieve high performance and flexibility [151].

9.2.4 Hardware Inference of Irregular Data Patterns. While fine-grained pruning can lead to high compression, it also produces data distribution irregularity, making conversion of

compression into speedups challenging [52, 55, 122]. For example, for AlexNet on GPUs with structured pruning, a compression ratio of 3.0 led to 3.0× greater throughput [74], while, in contrast, element-wise pruning resulted in superior compression (9.0×) but the same throughput [55]. In this context, there is an emerging need for hardware accelerators to support compressed and sparse networks to become competitive high-performance, low-power GPP alternatives. Works based on custom hardware, such as ESE [52] on FPGAs and Cnvlutin [4] and Minerva [114] on ASICs, featured fast and dynamic arithmetic operation avoidance suiting fine-grained pruning, achieving superior throughput and energy efficiency to GPP implementations. Future works should explore the further use of design flexibility to realise more acceleration from sparsity.

9.2.5 Parameter Hardening. Almost all works exploiting existing approximation still see the storage of parameters in DRAM for hardware reusability and scalability. With the large memory transfer reductions achievable through the use of aggressive methods including binarisation, logarithmic quantisation and weight sharing, however, smaller-sized parameters can fit on-chip more easily. It has thus become increasingly sensible to harden parameters into logic, reducing off-chip memory fetches. In some cases, memory fetching can be eliminated entirely. With base-two logarithmic quantisation, for example, multiplications are converted into binary shifts, which, when hardened, can be implemented without consuming any logic. Industrial firms such as Microsoft and Google have focussed their efforts on the optimisation of datacentre-scale DNN inference with custom ASIC [66] and FPGA [28] designs. Their huge throughput and energy efficiency requirements justify the use of extremely large and specialised accelerators employing loop unrolling and parameter hardening. Future research can explore the feasibility of this approach, showing how it trades off design reusability and scalability for throughput and efficiency.

10 SUMMARY

In this article, we discussed the past, present, and future of DNN approximation for custom hardware. With a roofline model analysis, we explained why DNNs' algorithmic advancement favours custom implementations, demonstrating how FPGAs and ASICs can offer performance superior to that of alternative platforms through the exploitation of approximation. With a comprehensive selection of state-of-the-art publications, we presented in-depth evaluations and comparisons of DNN approximation algorithms along with their respective hardware implementations. We summarised the current trends in the field, based on which we posed several research questions that are yet to be sufficiently answered. Through this work, we hope to inspire new and exciting developments in DNN approximation that tap into the full potential offered by custom hardware platforms.

REFERENCES

[1] Intel AI. 2017. Intel Nervana Neural Network Processors (NNP) Redefine AI Silicon. Retrieved from https://ai.intel.com/intel-nervana-neural-network-processors-nnp-redefine-ai-silicon/.

[2] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, and Gi-Joon Nam. 2015. TrueNorth: Design and tool flow of a 65mW 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput.-aided Design Integr. Circ. Syst.* 34, 10 (2015).

[3] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. 2017. Bit-pragmatic deep neural network computing. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.

[4] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie E. Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM SIGARCH Computer Architecture News*.

[5] Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. 2017. Ternary neural networks for resource-efficient AI applications. In *Proceedings of the International Joint Conference on Neural Networks*.

[6] Amjad Almahairi, Nicolas Ballas, Tim Cooijmans, Yin Zheng, Hugo Larochelle, and Aaron Courville. 2016. Dynamic capacity networks. In *Proceedings of the International Conference on Machine Learning*.

[7] Amara Amara, Frederic Amiel, and Thomas Ea. 2006. FPGA vs. ASIC for low power applications. *Microelectron. J.* 37, 8 (2006).

[8] Hesham Amin, K. Mervyn Curtis, and Barrie R. Hayes-Gill. 1997. Piecewise linear approximation applied to nonlinear function of a neural network. *IEE Proceedings—Circuits, Devices and Systems* 144, 6 (1997).

[9] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. 2018. YodaNN: An architecture for ultra-low power binary-weight CNN acceleration. *IEEE Trans. Comput.-aided Design Integr. Circ. Syst.* 37, 1 (2018).

[10] Sayed O. Ayat, Mohamed Khalil-Hani, and Ab Al-Hadi Ab Rahman. 2018. Optimizing FPGA-based CNN accelerator for energy efficiency with an extended roofline model. *Turkish J. Electric. Eng. Comput. Sci.* 26, 2 (2018).

[11] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *Proceedings of the Conference on Neural Information Processing Systems*.

[12] Nathan Bell and Michael Garland. 2009. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*.

[13] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional computation in neural networks for faster models. In *Proceedings of the International Conference on Learning Representations*.

[14] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[15] Andrew Boutros, Sadegh Yazdanshenas, and Vaughn Betz. 2018. Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs. In *Proceedings of the International Conference on Field-programmable Logic and Applications*.

[16] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep learning with low precision by half-wave Gaussian quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[17] Adrian Caulfield, Eric Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A cloud-scale acceleration architecture. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.

[18] Andre X. M. Chang and Eugenio Culurciello. 2017. Hardware accelerators for recurrent neural networks on FPGA. In *Proceedings of the International Symposium on Circuits and Systems*.

[19] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*.

[20] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. 2017. Learning efficient object detection models with knowledge distillation. In *Proceedings of the Conference on Neural Information Processing Systems*.

[21] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *Proceedings of the International Conference on Machine Learning*.

[22] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, and Ninghui Sun. 2014. DaDianNao: A machine-learning supercomputer. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.

[23] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-state Circuits* 52, 1 (2017).

[24] Jian Cheng, Peisong Wang, Gang Li, Qinghao Hu, and Hanqing Lu. 2018. Recent advances in efficient computation of deep convolutional neural networks. *Front. Info. Technol. Electron. Eng.* 19, 1 (2018).

[25] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Process. Mag.* 35, 1 (2018).

[26] Yu Cheng, Felix X. Yu, Rogerio S. Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the International Conference on Computer Vision*.

[27] Yu Cheng, Felix X. Yu, Rogerio S. Feris, Sanjiv Kumar, Alok Choudhary, and Shih-Fu Chang. 2015. Fast neural networks with circulant projections. *arXiv preprint arXiv:1502.03436* (2015).

[28] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengil, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Christian Boehn, Oren Firestein, Alessandro Forin, Kang S. Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Tamas Juhasz, Ratna K. Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Steve Reinhardt, Adam Sapek, Raja Seera, Balaji Sridharan, Lisa Woods, Phillip Yi-Xiao, Ritchie Zhao, and Doug Burger. 2017. Accelerating persistent neural networks at datacenter scale. In *Proceedings of the Conference on Hot Chips*.

[29] Philip Colangelo, Nasibeh Nasiri, Eriko Nurvitadhi, Asit Mishra, Martin Margala, and Kevin Nealis. 2018. Exploration of low numerical precision deep learning inference using Intel FPGAs. In *Proceedings of the International Symposium on Field-programmable Custom Computing Machines*.

[30] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv preprint arXiv:1602.02830* (2016).
[31] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Conference on Neural Information Processing Systems*.
[32] Matthieu Courbariaux, Jean-Pierre David, and Yoshua Bengio. 2015. Low precision storage for deep learning. In *International Conference on Learning Representations*.
[33] Vin De Silva and Lek-Heng Lim. 2006. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM J. Matrix Anal. Appl.* 30, 3 (2006).
[34] Wei Deng, Wotao Yin, and Yin Zhang. 2013. Group sparse optimization by alternating direction method. In *Proceedings of the International Society for Optical Engineering*.
[35] Misha Denil, Babak Shakibi, Laurent Dinh, and Nando De Freitas. 2013. Predicting parameters in deep learning. In *Proceedings of the Conference on Neural Information Processing Systems*.
[36] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the Conference on Neural Information Processing Systems*.
[37] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, and Geng Yuan. 2017. CirCNN: Accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.
[38] Caiwen Ding, Ao Ren, Geng Yuan, Xiaolong Ma, Jiayu Li, Ning Liu, Bo Yuan, and Yanzhi Wang. 2018. Structured weight matrices-based hardware accelerators in deep neural networks: FPGAs and ASICs. *arXiv preprint arXiv:1804.11239* (2018).
[39] Wlodzislaw Duch and Norbert Jankowski. 1999. Survey of neural transfer functions. *Neural Comput. Surveys* 2, 1 (1999).
[40] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *Proceedings of the IEEE Computer Society Computer Vision and Pattern Recognition Workshops*.
[41] Sean Fox, David Boland, and Philip H. W. Leong. 2018. FPGA FastFood – A high speed systolic implementation of a large scale online kernel method. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[42] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. 2017. Learning to fly by crashing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.
[43] Chang Gao, Daniel Neil, Enea Ceolini, Shih-Chii Liu, and Tobi Delbruck. 2018. DeltaRNN: A power-efficient recurrent neural network accelerator. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[44] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. 2018. ReBNet: Residual binarized neural network. In *Proceedings of the IEEE International Symposium on Field-programmable Custom Computing Machines*.
[45] Robert M. Gray. 2006. Toeplitz and circulant matrices: A review. *Found. Trends Commun. Info. Theory* 2, 3 (2006).
[46] Yijin Guan, Zhihang Yuan, Guangyu Sun, and Jason Cong. 2017. FPGA-based accelerator for long short-term memory recurrent neural networks. In *Proceedings of the Asia and South Pacific Design Automation Conference*.
[47] Denis A. Gudovskiy and Luca Rigazio. 2017. ShiftCNN: Generalized low-precision architecture for inference of convolutional neural networks. *arXiv preprint arXiv:1706.02393* (2017).
[48] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Song Yao, Song Han, Yu Wang, and Huazhong Yang. 2016. Angel-Eye: A complete design flow for mapping CNN onto customized hardware. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*.
[49] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2017. A survey of FPGA based neural network accelerator. *ACM Trans. Reconfig. Technol. Syst.* 9, 4 (2017).
[50] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient DNNs. In *Proceedings of the Conference on Neural Information Processing Systems*.
[51] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the International Conference on Machine Learning*.
[52] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, and Yu Wang. 2017. ESE: Efficient speech recognition engine with sparse LSTM on FPGA. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[53] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*.

[54] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proceedings of the International Conference on Learning Representations*.

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

[56] Babak Hassibi and David G. Stork. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Proceedings of the Conference on Neural Information Processing Systems*.

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

[58] Gopalakrishna Hegde and Nachiket Kapre. 2018. CaffePresso: Accelerating convolutional networks on embedded SoCs. *ACM Trans. Embed. Comput. Syst.* 17, 1 (2018).

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

[60] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[61] Intel. 2018. Intel at Hot Chips 2018: Showing the Ankle of Cascade Lake. Retrieved from https://www.anandtech.com/show/13239/intel-at-hot-chips-2018-showing-the-ankle-of-cascade-lake.

[62] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *arXiv preprint arXiv:1712.05877* (2017).

[63] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*.

[64] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011).

[65] Norman P. Jouppi, Cliff Young, Nishant Patil, and David Patterson. 2018. A domain-specific architecture for deep neural networks. *Commun. ACM* 61, 9 (2018).

[66] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, and Al Borchers. 2017. In-datacenter performance analysis of a Tensor Processing Unit. In *Proceedings of the International Symposium on Computer Architecture*.

[67] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.

[68] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *International Conference on Computer Vision*.

[69] Soroosh Khoram and Jing Li. 2018. Adaptive quantization of neural networks. In *Proceedings of the International Conference on Learning Representations*.

[70] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K. Bansal, William Constable, Oguz Elibol, Scott Gray, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Kloss Carey, Ruby J. Pai, and Naveen Rao. 2017. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Proceedings of the Conference on Neural Information Processing Systems*.

[71] Alexandros Kouris, Stylianos I. Venieris, and Christos-Savvas Bouganis. 2018. CascadeCNN: Pushing the performance limits of quantisation in convolutional neural networks. In *Proceedings of the International Conference on Field-programmable Logic and Applications*.

[72] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2017. Deep convolutional neural network inference with floating-point weights and fixed-point activations. In *Proceedings of the International Conference on Machine Learning*.

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

[74] Vadim Lebedev and Victor Lempitsky. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[75] Yann LeCun, John S. Denker, and Sara A. Solla. 1990. Optimal Brain Damage. In *Proceedings of the Conference on Neural Information Processing Systems*.

[76] Edward H. Lee, Daisuke Miyashita, Elaina Chai, Boris Murmann, and Simon S. Wong. 2017. LogNet: Energy-efficient neural networks using logarithmic computation. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*.

[77] Bing Li, Wei Wen, Jiachen Mao, Sicheng Li, Yiran Chen, and Hai Li. 2018. Running sparse and low-precision neural network: When algorithm meets hardware. In *Proceedings of the Asia and South Pacific Design Automation Conference*.

[78] Fengfu Li and Bin Liu. 2016. Ternary weight networks. In *Proceedings of the Conference on Neural Information Processing Systems*.
[79] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. 2017. Training quantized nets: A deeper understanding. In *Proceedings of the Conference on Neural Information Processing Systems*.
[80] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans P. Graf. 2017. Pruning filters for efficient convnets. In *Proceedings of the International Conference on Learning Representations*.
[81] Sicheng Li, Wei Wen, Yu Wang, Song Han, Yiran Chen, and Hai Li. 2017. An FPGA design framework for CNN sparsification and acceleration. In *Proceedings of the IEEE International Symposium on Field-programmable Custom Computing Machines*.
[82] Sicheng Li, Chunpeng Wu, Hai Li, Boxun Li, Yu Wang, and Qinru Qiu. 2015. FPGA acceleration of recurrent neural network based language model. In *Proceedings of the IEEE International Symposium on Field-programmable Custom Computing Machines*.
[83] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. 2018. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275, C (2018).
[84] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *Proceedings of the International Conference on Machine Learning*.
[85] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime neural pruning. In *Proceedings of the Conference on Neural Information Processing Systems*.
[86] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In *Proceedings of the Conference on Neural Information Processing Systems*.
[87] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2015. Neural networks with few multiplications. In *Proceedings of the International Conference on Learning Representations*.
[88] Ji Liu, Przemyslaw Musialski, Peter Wonka, and Jieping Ye. 2013. Tensor completion for estimating missing values in visual data. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 1 (2013).
[89] Lanlan Liu and Jia Deng. 2017. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *arXiv preprint arXiv:1701.00299* (2017).
[90] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*.
[91] Xuan Liu, Di Cao, and Kai Yu. 2018. Binarized LSTM language model. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*.
[92] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the International Conference on Computer Vision*.
[93] Zhiyun Lu, Vikas Sindhwani, and Tara N. Sainath. 2016. Learning compact recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*.
[94] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-Sun Seo. 2017. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[95] Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. 2017. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462* (2017).
[96] Asit Mishra, Eriko Nurvitadhi, Jeffrey J. Cook, and Debbie Marr. 2018. WRPN: Wide reduced-precision networks. In *Proceedings of the International Conference on Learning Representations*.
[97] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning convolutional neural networks for resource efficient inference. In *Proceedings of the International Conference on Learning Representations*.
[98] Alexander Monakov, Anton Lokhmotov, and Arutyun Avetisyan. 2010. Automatically tuning sparse matrix-vector multiplication for GPU architectures. In *Proceedings of the International Conference on High-performance Embedded Architectures and Compilers*.
[99] Bert Moons and Marian Verhelst. 2016. A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale convnets. In *Proceedings of the IEEE Symposium on VLSI Circuits*.
[100] Duncan Moss, Srivatsan Krishnan, Eriko Nurvitadhi, Piotr Ratuszniak, Chris Johnson, Jaewoong Sim, Asit Mishra, Debbie Marr, Suchit Subhaschandra, and Philip H. W. Leong. 2018. A customizable matrix multiplication framework for the Intel HARPv2 Xeon + FPGA platform. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[101] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. 2015. Adding gradient noise improves learning for very deep networks. In *Proceedings of the International Conference on Learning Representations*.

[102] Steven J. Nowlan and Geoffrey E. Hinton. 1992. Simplifying neural networks by soft weight-sharing. *Neural Comput.* 4, 4 (1992).

[103] Eriko Nurvitadhi, Jeff Cook, Asit Mishra, Debbie Marr, Kevin Nealis, Philip Colangelo, Andrew Ling, Davor Capalija, Utku Aydonat, Sergey Shumarayev, and Aravind Dasu. 2018. In-package domain-specific ASICs for Intel Stratix 10 FPGAs: A case study of accelerating deep learning using TensorTile ASIC. In *Proceedings of the International Conference on Field-programmable Logic and Applications*.

[104] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason O. G. Hock, Yeong T. Liew, Krishnan Srivatsan, Duncan Moss, and Suchit Subhaschandra. 2017. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.

[105] Nvidia. 2018. CUDA C Programming Guide. Retrieved from https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#arithmetic-instructions.

[106] Nvidia. 2018. NVIDIA Turing Architecture Whitepaper. Retrieved from https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[107] Georg Ofenbeck, Ruedi Steinmann, Victoria Caparros, Daniele G. Spampinato, and Markus Puschel. 2014. Applying the roofline model. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*.

[108] Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. 2016. Recurrent neural networks with limited numerical precision. *arXiv preprint arXiv:1608.06902* (2016).

[109] Thorbjörn Posewsky and Daniel Ziener. 2018. Throughput optimizations for FPGA-based deep neural network inference. *Microprocess. Microsyst.* 60 (2018).

[110] Adrien Prost-Boucle, Alban Bourge, Frédéric Pétrot, Hande Alemdar, Nicholas Caldwell, and Vincent Leroy. 2017. Scalable high-performance architecture for convolutional ternary neural networks on FPGA. In *Proceedings of the International Conference on Field-programmable Logic and Applications*.

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

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

[113] Mohammad S. Razlighi, Mohsen Imani, Farinaz Koushanfar, and Tajana Rosing. 2017. LookNN: Neural network with no multiplication. In *Proceedings of the Design, Automation and Test Conference in Europe*.

[114] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae-Kyu Lee, José M. Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ACM SIGARCH Computer Architecture News*.

[115] Michalis Rizakis, Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Approximate FPGA-based LSTMs under computation time constraints. In *Proceedings of the International Symposium on Applied Reconfigurable Computing*.

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

[117] Bita D. Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. 2016. Delight: Adding energy dimension to deep neural networks. In *Proceedings of the International Symposium on Low Power Electronics and Design*.

[118] Bita D. Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. 2017. Deep3: Leveraging three levels of parallelism for efficient deep learning. In *Proceedings of the Design Automation Conference*.

[119] Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. 2017. Analytical guarantees on numerical precision of deep neural networks. In *Proceedings of the International Conference on Machine Learning*.

[120] Mohammad Samragh, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2017. Customizing neural networks for efficient FPGA implementation. In *Proceedings of the IEEE International Symposium on Field-programmable Custom Computing Machines*.

[121] Eric Schurman and Jake Brutlag. 2009. The user and business impact of server delays, additional bytes, and HTTP chunking in Web search. In *Proceedings of the Velocity Conference*.

[122] Abigail See, Minh-Thang Luong, and Christopher D. Manning. 2016. Compression of neural machine translation models via pruning. In *Proceedings of the SIGNLL Conference on Computational Natural Language Learning*.

[123] Sayeh Sharify, Alberto Delmás, Kevin Siu, Patrick Judd, and Andreas Moshovos. 2018. Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks. In *Proceedings of the Design Automation Conference*.

[124] Sayeh Sharify, Mostafa Mahmoud, Alberto Delmás, Milos Nikolic, and Andreas Moshovos. 2018. Laconic deep learning computing. *arXiv preprint arXiv:1805.04513* (2018).
[125] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon K. Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.
[126] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *Proceedings of the International Symposium on Computer Architecture*.
[127] Junzhong Shen, You Huang, Zelong Wang, Yuran Qiao, Mei Wen, and Chunyuan Zhang. 2018. Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[128] Sungho Shin, Yoonho Boo, and Wonyong Sung. 2017. Fixed-point optimization of deep neural networks with adaptive step size retraining. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*.
[129] Sungho Shin, Kyuyeon Hwang, and Wonyong Sung. 2016. Fixed-point performance analysis of recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*.
[130] Nathan Silberman and Sergio Guadarrama. 2016. TensorFlow-Slim Image Classification Model Library. Retrieved from https://github.com/tensorflow/models/tree/master/research/slim.
[131] Vikas Sindhwani, Tara N. Sainath, and Sanjiv Kumar. 2015. Structured transforms for small-footprint deep learning. In *Proceedings of the Conference on Neural Information Processing Systems*.
[132] Suraj Srinivas and R. Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149* (2015).
[133] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014).
[134] Jiang Su, Julian Faraone, Junyi Liu, Yiren Zhao, David B. Thomas, Philip H. W. Leong, and Peter Y. K. Cheung. 2018. Redundancy-reduced MobileNet acceleration on reconfigurable logic for ImageNet classification. In *Proceedings of the International Symposium on Applied Reconfigurable Computing*.
[135] Wonyong Sung and Ki-Il Kum. 1995. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Trans. Signal Process.* 43, 12 (1995).
[136] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017).
[137] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. 2017. Inception-v4, Inception-ResNet and the impact of residual connections on learning. In *Proceedings of the Association for the Advancement of Artificial Intelligence*.
[138] Cheng Tai, Tong Xiao, Yi Zhang, and Xiaogang Wang. 2016. Convolutional neural networks with low-rank regularization. In *Proceedings of the International Conference on Learning Representations*.
[139] Wei Tang, Gang Hua, and Liang Wang. 2017. How to train a compact binary neural network with high accuracy? In *Proceedings of the Association for the Advancement of Artificial Intelligence*.
[140] Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. In *Proceedings of the International Conference on Learning Representations*.
[141] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip H. W. Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[142] Stylianos I. Venieris and Christos-Savvas Bouganis. 2016. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *Proceedings of the IEEE International Symposium on Field-programmable Custom Computing Machines*.
[143] Stylianos I. Venieris and Christos-Savvas Bouganis. 2017. Latency-driven design for FPGA-based convolutional neural networks. In *Proceedings of the International Conference on Field-programmable Logic and Applications*.
[144] Erwei Wang, James J. Davis, and Peter Y. K. Cheung. 2018. A PYNQ-based framework for rapid CNN prototyping. In *Proceedings of the IEEE International Symposium on Field-programmable Custom Computing Machines*.
[145] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, Yanzhi Wang, and Yun Liang. 2018. C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*.
[146] Zhisheng Wang, Jun Lin, and Zhongfeng Wang. 2017. Accelerating recurrent neural networks: A memory-efficient approach. *IEEE Trans. VLSI Syst.* 25, 10 (2017).
[147] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Proceedings of the Conference on Neural Information Processing Systems*.

[148] Darrell Williamson. 1991. Dynamically scaled fixed point arithmetic. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference.*

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

[150] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. 2018. Training and inference with integers in deep neural networks. In *Proceedings of the International Conference on Learning Representations.*

[151] Xilinx. 2018. Versal, the First Adaptive Compute Acceleration Platform. Retrieved from https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf.

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

[153] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision.*

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

[155] Chen Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2016. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In *Proceedings of the International Conference On Computer Aided Design.*

[156] Jialiang Zhang and Jing Li. 2018. PQ-CNN: Accelerating product quantized convolutional neural network on FPGA. In *Proceedings of the International Symposium on Field-programmable Custom Computing Machines.*

[157] Xiaofan Zhang, Xinheng Liu, Anand Ramachandran, Chuanhao Zhuge, Shibin Tang, Peng Ouyang, Zuofu Cheng, Kyle Rupnow, and Deming Chen. 2017. High-performance video content recognition with long-term recurrent convolutional network for FPGA. In *Proceedings of the International Conference on Field-programmable Logic and Applications.*

[158] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating binarized convolutional neural networks with software-programmable FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays.*

[159] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2016. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *Proceedings of the International Conference on Learning Representations.*

[160] Hao Zhou, Jose M. Alvarez, and Fatih Porikli. 2016. Less is more: Towards compact CNNs. In *Proceedings of the European Conference on Computer Vision.*

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

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

[163] Shilin Zhu, Xin Dong, and Hao Su. 2018. Binary ensemble neural network: More bits per network or more networks per bit? *arXiv preprint arXiv:1806.07550* (2018).

Received September 2018; revised December 2018; accepted January 2019