


REVIEW

Accelerating Deep Neural Networks implementation: A survey

Meriam Dhouibi  | Ahmed Karim Ben Salem | Afef Saidi | Slim Ben Saoud

Advanced Systems Laboratory, Tunisia Polytechnic School, University of Carthage, BP 743 La Marsa, 2078 Tunisia

Correspondence

Meriam Dhouibi, Advanced Systems Laboratory, Tunisia Polytechnic School, University of Carthage, BP 743, 2078, La Marsa, Tunisia,
Email: meriam.dhouibi@ept.rnu.tn

Abstract

Recently, Deep Learning (DL) applications are getting more and more involved in different fields. Deploying such Deep Neural Networks (DNN) on embedded devices is still a challenging task considering the massive requirement of computation and storage. Given that the number of operations and parameters increases with the complexity of the model architecture, the performance will strongly depend on the hardware target resources and basically the memory footprint of the accelerator. Recent research studies have discussed the benefit of implementing some complex DL applications based on different models and platforms. However, it is necessary to guarantee the best performance when designing hardware accelerators for DL applications to run at full speed, despite the constraints of low power, high accuracy and throughput. Field Programmable Gate Arrays (FPGAs) are promising platforms for the deployment of large-scale DNN which seek to reach a balance between the above objectives. Besides, the growing complexity of DL models has made researches think about applying optimization techniques to make them more hardware-friendly. Herein, DL concept is presented. Then, a detailed description of different optimization techniques used in recent research works is explored. Finally, a survey of research works aiming to accelerate the implementation of DNN models on FPGAs is provided.

1 | INTRODUCTION

Recently, DL technology has been used successfully for a variety of tasks in several fields of applications related to signal, information and image processing such as computer vision [1], Natural Language Processing (NLP) [2], medical [3], video games [4] and all areas of science and human activity. DL models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) continue to make great progress in solving complex problems. However, the deployment of such models is a hard task considering the massive amount of computation and the big storage requirements. Therefore, the performance of the model depends on the target hardware resources. The training and the inference phases of DL models are being executed on powerful computation machines using advanced technologies such as new multicore Central Processing Unit (CPU), Graphics Processing Unit (GPU) or clusters of CPUs and GPUs. Usually, GPU platforms are better on supporting training and inference of more sophisticated models. GPU technology offers a high computation capacity but ensures the interdependence of the

data which is expensive in terms of power. Application Specific Integrated Circuits (ASICs) can achieve even higher performance and can improve the energy efficiency, which is a key factor in embedded systems. However, the deployment of DL model on a customised ASIC requires high investments due to a long and complex design cycle. Recently, FPGAs become a promising solution to accelerate inference, they offer the performance advantages of reconfigurable logics with the high degree of flexibility. Specific hardware design on such platforms could be more efficient in speed and energy compared to other platforms. Moreover, the deployment of large-scale DNNs with large numbers of parameters is still a daunting task, because the large dimensionality of such models increases the computation and data movement. So, to deploy such sophisticated models in embedded platforms and to obtain a more robust model, the internal operations and number of parameters can be reduced by optimising the network architecture. Several optimizations techniques were discussed in literature. One of the most popular optimization approaches that makes models faster, energy efficient and more hardware friendly is model compression which includes the low data

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2021 The Authors. *IET Computers & Digital Techniques* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

precision, pruning network, low-rank approximation, etc. Furthermore, for efficient implementation of an optimised DL model, further acceleration improvement is required. Indeed, it is necessary to maximise the utilization of all offered opportunities at several levels of hardware/software codesign to achieve high performance in terms of precision, energy consumption and throughput. This survey takes a deep dive into DL implementation on advanced and dedicated computation platforms and reveals its bottlenecks. In addition, it is focusing on hardware and software techniques to optimise the implementation of DNNs and also provides a summary of recent research work. There are some surveys that have been published dealing with DL implementation. However, those papers have not discussed the state of the art in different hardware platforms. Most of the recent surveys have focussed on FPGA-based CNN acceleration without pointing out the choice of FPGA over other platforms. Another strong aspect of our work is that we discussed the optimization of DNNs in both levels' software and hardware. Furthermore, we presented a classification of advanced hardware acceleration techniques based on throughput and energy optimizations. An investigation of the algorithmic side and its effect on designing accelerators is also included in this survey. Additionally, we exposed the tools that can automatically generate hardware design from software that are used for implementing and evaluating deep learning approaches. Herein,

- Section 2 presents the basics of DL and its popular models and architectures currently in use and turns the lights on the complexity of these models.
- Section 3 describes the various hardware platforms used to implement DNNs.
- Section 4 exposes the optimization techniques that can be applied to make the model more efficient in terms of speed and power.

Finally, synthesis of different acceleration techniques explored in recent research works is given and analysed.

2 | BACKGROUND AND MOTIVATIONS

Currently, DL represents the leading-edge solution in virtually all relevant machine learning tasks in a large variety of fields [5,6]. DL algorithms are showing significant improvement over traditional machine learning algorithms based on manual extraction of relevant features (handcrafted features) [7]. DL models perform a hierarchical feature extraction and show also better performance with the increase of the amount of data [8]. There are different methods and architectures of DL such as Multi-Layer Perceptron (MLP), Autoencoder (AE), Deep Belief Network (DBN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) including Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), Generative Adversarial Network (GAN), Deep Reinforcement Learning (DRL), etc. [9]. These models have covered several fields with a variety of applications. Particularly,

CNN models have demonstrated impressive performance in computer vision applications such as autonomous car vision systems [10], drone navigation, robotics [11], etc. CNNs, have also proved to be more effective in medical field and specially in image recognition. It have been adopted at detecting a tumour or any other type of lesion than the most experienced radiologists [12]. In Ref. [13], an image extracted from Magnetic Resonance Imaging (MRI) of a human brain was processed to predict Alzheimer's disease using CNN. DL models are also used in drug research by predicting molecular properties such as toxicity or binding capacity. In particular, DL can be used to simulate biological or chemical processes of different molecules without the need for expensive software simulators and is 30,000 times faster [14]. Moreover, RNN models have exiled in natural language processing including automatic speech recognition, recommendation systems, audio recognition, machine translation, social media filtering, etc. For example, various LSTM models have been proposed for sequence to sequence mapping that are suitable for machine translation [15]. Furthermore, CNNs and RNNs were combined to add sounds to silent movies [16] and to generate captions that describes the contents of images [17]. Besides, it is important to note that the effective implementation of DL models on embedded platforms is behind this diffusion of such applications. The performance of such AI algorithms using DL models lies on the capacity of processors in supporting the DNN with its varied number of layers, neurons per layer, multiple filters, filter sizes and channels while treating large dataset. Indeed, DL workloads are both computation and memory intensive. For example, the well-known CNN network ResNet50 [18] requires up to 7.7 billion floating point operations (FLOPs) and 25.6 million model parameters to classify a $224 \times 224 \times 3$ image. As shown in Figure 1, the complex and larger model VGG16 [19] with 138.3 million parameters model size, requires up to 30.97 Giga FLOPs (GFLOPs). Thus, the number of operations and parameters increases with the complexity of the model architecture. Table 1 presents the state-of-the-art models' sizes and complexities (Table 2).

VGG models were developed by the Visual Geometry Group from University of Oxford and are the most preferred choices in the community for extracting features from images. They are widely used in many applications despite the expensive architecture in both terms of parameters number and computational requirements (Figure 1). The large dimensionality of these models increases the computation and data movement. More precisely, it increases the amount of generated data which its movement considered more expensive than computation, in terms of power on hardware platforms [21].

At this inflection point, it is therefore necessary to benefit from new design methodology, to make good use of new design opportunities and to explore some optimization techniques to reduce the network size and to enhance the implementation performance in terms of throughput and energy consumption. Besides, the choice of suitable hardware platform to implement a DL model is of paramount importance [24]. In the next section, we will explore different computation platforms of DL implementation.

FIGURE 1 Computational cost of most popular models: inference on ImageNet dataset [20]

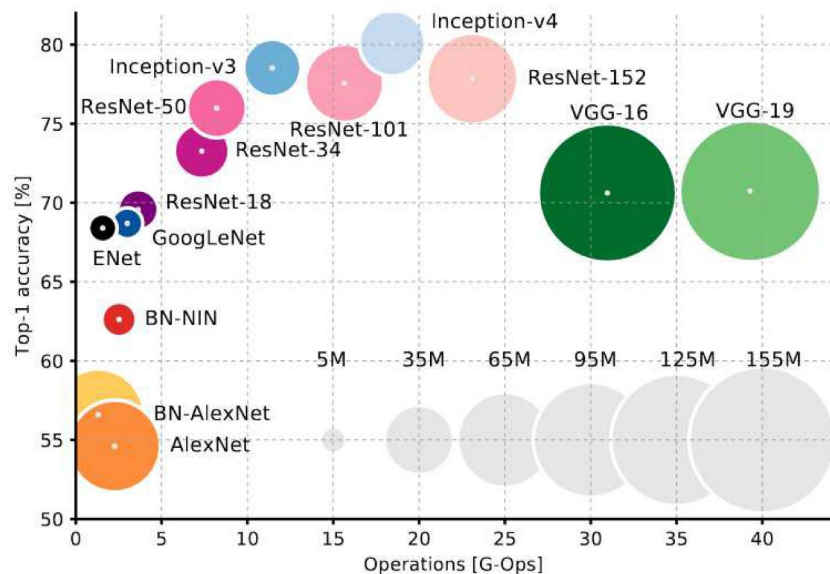


TABLE 1 Size and complexity of state-of-the-art models

Model	AlexNet [22]	VGG [19]		ResNet [18]		GoogLeNet [23]
		VGG16	VGG19	ResNet50	ResNet152	
Operations(GFLOPs)	1.4	30.97	39	7.7	22.6	1.57
Parameters(M)	58.3	138.3	144	25.6	57	6

3 | COMPUTATION PLATFORM OF DL IMPLEMENTATION

The employment of DL into daily applications of different fields will depend on the ease with which it will be possible to deploy DL model on small, low-power devices rather than large servers. In majority of cases, the training phase is performed in the cloud. However, the inference phase is less demanding, it can happen locally or in the cloud depending on the application [24]. Research is underway on the two phases implementation using parallel architectures on different hardware targets and computing devices. Four major types of technology are being used to accelerate DNNs: CPU, GPU, FPGA and ASIC.

3.1 | Central processing units

Traditionally, DNNs were mainly tested on the CPU of a computer. The CPU works by sequentially performing the computations that are sent to it. Sometimes, a programme has different tasks that can be calculated independently of each other. To optimise the time required to complete all tasks, many processors have multiple threads or cores that can perform parallel calculations. Some manufacturers have sought to optimise the hardware architectures of their processors to

meet the needs of DL: Intel has tweaked the CPUs of its servers to improve its performance with DL [25]. Google has developed a chip to perform DL tasks more economically [26]. However, it is still very difficult for CPUs, even with multicore architecture to support the high computation and the storage complexity of large DNN models.

3.2 | Graphics processing units

A GPU excels in parallel computing. CPU has typically between one and eight cores, and high-end GPUs have thousands of cores (e.g. GeForce GTX TITAN Z included 5760 cores, the last one is GeForce RTX 2080...). GPUs are slow during sequential operations, but shine when given tasks that can run in parallel. Since the operations required to run a DL algorithm can be done in parallel, GPUs became extremely valuable tools. Furthermore, by using OpenCL [27], an open standard for portable parallelisation, compute kernels written using a limited subset of the C programming language can be launched on GPUs. In this perspective, NVIDIA has invested much in its CUDA (Compute Unified Device Architecture) language to make it support the most DL development frameworks. Similar to OpenCL CUDA affords an environment of general-purpose programming and enables parallel processing over NVIDIA GPU's cores. NVIDIA GPUs are currently the most used for

TABLE 2 Reduce precision effect on DNN models

Reduce Precision technique	Reference	DL Model	Bitwidth				Results	
			Input	Weight	Activation	Gradient	Float 32-Bit Baseline Top-1 Accuracy	After Reduction Accuracy Loss
Reduce weight	[49]	MobileNetV1	-	8-bit	32-bit	32-bit	70.77%	2.74%
	INQ [50]	ResNet-18	-	5-bit	32-bit	32-bit	68.27%	-0.71%
			-	4-bit	32-bit	32-bit		-0.62%
			-	3-bit	32-bit	32-bit		0.19%
			-	2-bit	32-bit	32-bit		2.25%
	TWN [51]	ResNet-18	-	(ternary)		32-bit	68.27%	6.47%
			-	1-bit	32-bit	32-bit	68.27%	7.47%
Reduce weight and activation	BWNH [64]	ResNet-18	-	1-bit	32-bit	32-bit	68.27%	3.97%
	FFN [65]	AlexNet	-	2-bit	32-bit	32-bit	57.20%	1.70%
	Ristretto [53]	CaffeNet	-	8-bit	8-bit	32-bit	56.90%	0.90%
	Balanced quantization [54]	GoogLeNet	-	8-bit	8-bit	32-bit	71.50%	4.90%
			-	4-bit	4-bit	32-bit		3.80%
	QNN [55]	GoogLeNet	-	4-bit	4-bit	32-bit	71.50%	5.10%
	HWGQ [56]	GoogLeNet	-	1-bit	2-bit	32-bit	68.70%	5.70%
Reduce input, weight and activation	BNN [57]	AlexNet	1-bit	1-bit	1-bit	32-bit	57.20%	30.10%
	XNOR-Net [52]	AlexNet	1-bit	1-bit	1-bit	32-bit	56.60%	12.40%
Reduce weight, activation and gradient	DoReFa-Net [58]	AlexNet	-	8-bit	8-bit	8-bit	55.90%	2.90%
			-	1-bit	4-bit	6-bit		7.70%
			-	1-bit	3-bit	6-bit		8.80%
			-	1-bit	2-bit	8-bit		9.60%
			-	1-bit	2-bit	6-bit		9.80%
			-	1-bit	1-bit	8-bit		16.40%
			-	1-bit	1-bit	6-bit		16.40%

implementing DL algorithms. Most lately, NVIDIA [28] invented NVDLA, a scalable and highly configurable open source accelerator for DL inference to simplify integration and portability. In late 2018, AMD announced the first 7 nm (nanometer) GPU specifically designed for DL. The company's new Radeon deliver up to 7.4 TFLOPS (one trillion floating point operations per second). AMD revealed also a software to improve performance [29]. This proves the interest shown by manufacturers in choosing the right hardware that best suits the deployment of these DNNs. The Nvidia Tesla V100 for example, embeds 640 hearts 'Tensor'. These units offer neural networks a high computing capacity of over 100 teraflops and are particularly suited to popular development frameworks.

3.3 | FPGA

When evaluating the acceleration of hardware platforms, the trade-off between flexibility and performance must inevitably be taken into consideration. FPGAs serve as a good compromise between flexibility and performance. They are a reconfigurable integrated circuit with programmable processor cores. They offer the performance advantages of integrated circuits, with the high degree of flexibility. At a low level, FPGAs can implement sequential logic using Flip-Flops (FFs) and combinational logic using Look-Up Tables (LUTs). FPGAs also contain hardened components for functions that are commonly used, such as full processor cores,

communication cores, arithmetic cores, and RAM blocks. In addition, the adaptation of the System-on-Chip (SoC) design approach, in which the ARM coprocessors and FPGAs logic cells are generally located on the same chip have enhanced the flexibility of such devices. The FPGAs current market is dominated by Intel (nee Altera) and Xilinx, representing a combined market share of 85% [30]. On FPGAs, programmable logic cells can be used to implement the data and control path. They are also able to exploit the distributed memory on chip, and the pipelines parallelism, which are naturally part of the methods of deep feed-forward networks. FPGAs also support partial dynamic reconfiguration, which may have implications for large DL models, where individual layers could be reconfigured on the FPGA without disrupting the current calculation in the other layers. To speed up hardware designs FPGA platforms could be a promising perspective compared to GPUs. With fixed architectures like GPUs, a software execution model is tracked and structured around the execution of tasks in parallel on independent computing units: the goal of developing DL techniques for GPUs is to adapt the algorithms to follow this architecture, where the computation is carried out in parallel and where the interdependence of the data is ensured. However, when developing DL techniques for FPGAs, it is less important to adapt algorithms for a fixed computation structure, which allows more flexibility to explore algorithm optimizations. Techniques that require many complex low-level hardware control operations that are difficult to implement in high-level software languages are of particular interest for FPGA implementations. Recently, software-level programming models for FPGA have been adopted, including OpenCL, High-level synthesis (HLS), C, C++, making it a more attractive option [31]. In this perspective, Xilinx invented PYNQ to design embedded systems with their Zynq SoCs on easier way. It uses the Python language and libraries, which offers the benefits of programmable logic and microprocessors in Zynq that help building high performance embedded DL applications [32,33]. Furthermore, to accelerate DL inference with optimised and tuned hardware and software, Xilinx unveiled an adaptive compute acceleration platform (ACAP), Versal [34] a new heterogeneous compute architecture. Versal delivered higher performance (8×) than high-end GPUs. More recently, Xilinx designed an integrated IP block for Zynq SoC and MPSoC devices which is a programmable engine dedicated for CNNs called DL Processor Unit (DPU) [35]. Lately, FPGA-based accelerators like Xilinx Alveo cards [36] with new architecture appeared more often. They offer FPGAs ready to programme on the accelerator cards which can be directly plugged into servers and allow reconfigurable acceleration to adapt to continuous optimization of DL algorithms. For example, when executing inference, the Alveo U250 reduces latency by 3× over GPUs. Another FPGA-based multi-accelerator platform, Maxeler's MPC-X 2000 [37], that supports reconfigurable designs is widely used. It is comprised with Data Flow Engines (DFE) each using a Xilinx Virtex-6 FPGA. Currently, the Cloud represents a simple and efficient solution for using FPGAs without investing in specific hardware. In major cloud platforms and modern data centres, FPGA based

accelerators have showed impressive performance in terms of parallel computing and power consumption. Microsoft Azure cloud computing platform integrates Altera Stratix FPGA [38]. Amazon AWS provided Elastic Compute Cloud (EC2) F1 [39], a compute instance based to accelerate data centre workloads including DL inference. Equipped with eight Virtex UltraScale + VU9P FPGAs, F1 instance can perform up to 170 TOPs (tera operation per second) with INT8 data representation. An FPGA computing instance provides an easy way to create FPGA design with dedicated and customised hardware accelerators, based on the cloud elastic computing Framework. Alibaba Cloud [40], Huawei Cloud [41], Tencent Cloud [42], Baidu Cloud [43] and many others have also launched FPGA services. Alibaba Cloud's F1 instance is based on Intel Arrira10 GX 1150 computing card. The instance introduced by Tencent cloud is based on Xilinx Kintex UltraScale KU115 FPGA. This progress on the hardware devices side goes hand in hand with progress on software side. To make FPGAs much easier to use, Xilinx provided Vitis Unified Software Platform which is a development environment to design and deploy accelerated applications on Xilinx platforms such as ACAP, FPGA-instances in the cloud, Alveo cards and embedded platforms. Vitis AI, which is an integral part of Vitis that allows the acceleration of DL applications. It supports Tensorflow and Caffe frameworks and provides tools and APIs to optimise pre-trained DL models by applying pruning and quantization techniques. With specific designed hardware, FPGA exceeds GPU not only in energy efficiency but also in speed.

3.4 | ASIC

ASICs are designed for a specific fixed functionality or application. During its operating life, a customised ASIC has a fixed logic function because its digital circuitry is made up of gates and flip-flops permanently connected in silicon. Several research works have focussed on building customised ASICs to accelerate DL model training and inference [44]. Compared to FPGA, ASIC platforms with a customised architecture are more efficient in terms of power and speed. An ASIC can perform fixed operations extremely fast since the entire chip's logic area can be devoted to a set of narrow functions. Despite its high performance, designing an ASIC can be highly expensive due to its construction process complexity. A customised ASIC needs verification and frequent updates to keep abreast of new techniques. Moreover, the rapid evolving of DL modes requires design changing which is costly in terms of time and price for ASICs. Even with a lack of flexibility, ASICs are still an attractive solution for dealing efficiently with the massive workloads of DL models. Currently, more than 100 companies are building ASICs targets towards DL applications including Google, Facebook, etc. Google designed Tensor Processing Unit (TPU), a 28 nm customised ASIC to accelerate DL applications. The 700 MHz TPU performed 95 TFLOPs and 23 TFLOPs for 8-bit and 16-bit calculations respectively, whilst drawing only 40W. TPU v2, announced in May 2017, is a four ASIC board that can do 180 TFLOPs of performance. A

year later, Google announced TPU v3 and improves the peak performance to 420 TFLOPs. In February 2018, cloud TPUs that powers Google products like Translate, Search, Assistant, and Gmail became available for use in Google Cloud Platform (GCP) [45]. TPU can handle both training inference and it has the highest training throughput. More recently, Habana Labs startup developed the HL-1000 a 16 nm custom ASIC chip [46]. The designed architecture is very similar to that of Google's TPU using a large on-chip Static Random Access Memory (SRAM) and large matrix-multiply accelerator. The only difference is that Habana includes eight programable CPU cores to handle non convolutional layers, whereas Google implements these layers in fixed-function logic. The startup Gyr Falcon Technology Inc (GTI) [47] introduced Lightspeur 2801S, Lightspeur 2802M and Lightspeur 2803S edge-based ASICs for the deployment of AI application. Lightspeur 2801S, a 28 nm neural accelerator with no external memory and 28,000 parallel computing cores performs up to 2.8 TOPs and 9.3 TOPs/W. Based on 16-chip server, a 2803S performs 271 TOPs at 28W [48]. ASICs are still more efficient than FPGAs. However, the combination of GPUs training performance and FPGAs efficiency and flexibility for inference can be an alternative and promising solution.

While running DNNs, it is still difficult for CPUs to achieve high performance levels compared to GPUs, FPGAs and ASICs due to the massive computation and memory bandwidth requirements. However, GPUs with their high memory bandwidth and throughput are the most widely used for training DNNs. GPUs' high performance is due to their parallel processing. However, they consume a large amount of power. FPGAs and ASICs can also offer very high bandwidth by being directly connected to inputs. Moreover, compared with GPUs, FPGAs and ASICs can provide higher performance with lower power consumption while running DL algorithm. As DL models rapidly evolve and change, FPGAs offer more flexibility and reconfigurability than ASICs. Additionally, FPGAs are using new tools that make programing DNNs applications much easier.

For further improvement of performance, various optimization techniques have been proposed. In the next section, we give an overview of some of the most used techniques.

4 | OPTIMIZATION TECHNIQUES

There are several techniques focussed on modifying DL algorithms to make them more hardware-friendly with minimal loss of accuracy. Many approaches have been explored to effectively digest the redundancy of models and provide improved computing efficiency such as the low data precision, pruning network and Low-Rank Approximation (LRA).

4.1 | Precision reduction

The use of lower precision in representing data to run DNNs reduces the storage demand of the DNN models lowering the

data bandwidth requirements. It optimises the computing efficiency and improves performance. However, special attention must be paid to the possible degradation of accuracy. From the algorithmic perspective, recent research work can be divided into three categories: weights precision reduction, precision reduction of both weights and activations and precision reduction of inputs, weights and activations. Many researchers targeted weights precision reduction, since weights can reduce directly the network size. In Ref. [49], a friendly quantization applied on MobileNetV1 model, reached an accuracy of 68.03% in 8-bit weights representation, which almost closed the gap to the float point representation (70.77%). Zhou et al. presented INQ [50], a generalized quantization framework to convert any pre-trained full-precision CNN model with 32 bit floating point into a lossless low-precision version of weights with 5-bit, 4-bit, 3-bit and even 2-bit. The use of this framework on ResNet-18 improved accuracy for 5-bit and 4-bit quantization by 0.71% and 0.62% respectively. Li et al. squeezed the representation to 2-bit in Ref. [51] which resulted in 6.47% accuracy degradation. Also, Rastegari et al. proposed a fully binarised neural networks called BWN in Ref. [52]. BWN gained 32× memory saving with 12.4% accuracy degradation. Another recent research works, applied the precision reduction technique on weights and activations. Indeed, in Ref. [53], CaffeNet inference is successfully performed with 8-bit fixed-point representation of weights and activations and resulting in less than 1% degradation of accuracy. A Balanced Quantization method is introduced in Ref. [54]. It performed 66.6% top-1 accuracy when applying 8-bit representation of weights and activations on GoogLeNet which is less than 5% degradation compared to the float 32-bit baseline. Moreover, a quantized version of GoogLeNet with 4-bit weights and activations in Ref. [55] achieved 66.5% top-1 accuracy which is 5.1% drop in accuracy. Cai et al. [56] introduced Half-Wave Gaussian Quantization (HWGQ), that reduced the precision by 5.7% on GoogLeNet with binary weights and ternary activations. Some other researches have shown that quantized inputs, weights and activations can achieve better computational efficiency. The binarisation of inputs weights and activation is explored in Ref. [57]. The authors proposed a fully Binarised Neural Networks (BNN) that drastically reduced memory size and accesses. Based on BWN, Rastegari et al. [52] presented XNOR-Net by binarising all activations resulting in 58× faster convolutional operations. XNOR-Net performed better accuracy than BNN [57]. In Ref. [58], the authors proposed DoReFa-Net, a method that used low bitwidth parameter gradients to train CNN with low bitwidth inputs, weights and activations. DoReFa-Net performed comparable accuracy as the 32-bit baseline on SVHN and ImageNet datasets. Detailed results are summarised in Table 2. From the hardware perspective, a lot of work applied fixed point representation to implement DNNs and substantially reduced the bitwidth for energy and area savings, and throughput increasing. In Ref. [59], LSTM models (Google LSTM and Small LSTM) with 16-bit fixed-point data type were implemented on two FPGA platforms resulting in only 1.23% precision degradation. Moss et al. presented an FPGA-based customisable matrix

multiplication framework to run DNNs [60]. It allowed the runtime switching between static-precision bit-parallel and dynamic-precision bit-serial MAC (Multiply and Accumulate) implementations. The experimental results on AlexNet, VGGNet and ResNet reached up $50\times$ throughput increases versus FP32 baselines. In Ref. [61], the authors implemented Google LSTM in Xilinx FPGA using 12-bit fixed point which resulting better performance and only 0.3% precision degradation. In Ref. [62], Shen et al. implemented VGG-16 and C3D across multiple FPGA platforms with DSPs (Digital Signal Processor) that supports one 16-bit fixed-point multiply and add. It achieved an end-to-end performance $7.3\times$ better than software implementation. Following the same strategy, Zhang et al. [63] achieved a $3.1\times$ throughput speedup with the implementation of a long-term recurrent convolutional network LRCN on a Xilinx FPGA using a fixed-point quantization. Although, the use of this technique offers a substantial gain in throughput and energy efficiency. But less than 8 bits representation of the data values in large DNNs can increase the accuracy degradation (Table 4).

4.2 | Pruning

Neural networks are considered over-parametrised, as there is a large amount of a redundant parameters that are with small influence on the accuracy, which costs in computation as well in memory footprint. These parameters can be removed through a process called pruning which is often followed by some fine tuning to improve the accuracy. Recently, several research studies [75,76] have shown the effectiveness of this technique on model size reduction, the computations amount and indirectly the energy consumption with minimal accuracy degradation. There are many pruning methods in terms of weights, filters channels and feature maps. The core idea of weight pruning is to remove the redundancy of some weight by setting them to zero. Rather than searching exhaustively for the weights to be pruned per layer, Ref. [66] explored a technique to find automatically the possible pruned weights sets while minimising the loss over all weights. The test error of this method on ResNet110 and ResNet56 was respectively 6.50% and 6.67%. To guarantee the weight reduction ratio, Zhang et al. [67] proposed a systematic framework for weight pruning of DNN based on the alternating direction method of multipliers (ADMM). This approach achieves weight reduction on LeNet-5 and AlexNet models respectively with $71.2\times$ and $21\times$ with and no accuracy loss. Yang et al. [68] proposed the Energy-Aware Pruning (EAP) technique for weight pruning using the energy consumption estimation of CNN. This method achieves an energy consumption reduction for GoogLeNet and AlexNet respectively by $1.6\times$ and $3.7\times$, compared to their original models with less than 1% top-5 accuracy loss. For filter pruning, the basic idea is to remove the unimportant filters by an estimation of the filter's importance. Li et al. [69] reported a methodology to prune whole filters and their related feature maps by using as a measurement of filter importance the sum of the absolute values of filters. This approach reduced the

inference cost of VGG-16 and ResNet-110 respectively by 34%, 38% while maintaining nearly the original accuracy. The study by Huang et al. [70] suggested a 'try-and-learn' learning algorithm to prune filters in CNN while maintaining the performance. The proposed algorithm removes 63.7% redundant filters in FCN-32s and accelerated the inference by 37.0% on GPU and 49.1% on CPU. Recently, a new method for filter pruning was explored in Ref. [71] which is based on the sparsity induction of weights. The proposed technique achieves FLOPs reduction on VGG-16 on two datasets CIFAR10 and GTSRB respectively by 90.50% and 96.6% without accuracy loss. Channel pruning reduces the model size by removing the channels and the related filters as well as the corresponding feature maps. Several channel pruning methods were proposed, for instance, Ref. [72] investigated a method for channel selection called Discrimination-aware Channel Pruning (DCP). Experiments of this method on ResNet-50 showed that with 30% reduction of channels it outperforms several state-of-the-art methods by 0.39% in top-1 accuracy. The study by Liu and Wu [73] proposed a new channel pruning criterion based on the mean gradient of feature maps which reduces effectively the network FLOPs. Using this approach on VGG-16 and ResNet-110 achieves respectively $5.64\times$ and $2.48\times$ reduction in FLOPs, with less than 1% and 0.08% decrease in accuracy, respectively. Liu et al. [74] enforced a scaling factor during the training for channel pruning. The effectiveness of this approach was evaluated with several CNN models (VGGNet, ResNet and DenseNet). For VGGNet, it achieves $20\times$ reduction in model size and $5\times$ reduction in computing operations. More details are presented in Table 2. To achieve speedup, pruning can be combined with other techniques used for optimization. The work in Ref. [77] investigated the benefits and costs of quantization and pruning as well as the combination of the both. The evaluation of the approach on NVIDIA Jetson TX2 showed that when using pruning, the inference time and energy consumption was reduced, respectively by 28% and 22.5% with little saving in storage size. However, when using quantization, the model storage size was reduced by 75% while the inference time and energy was reduced respectively by $1.41\times$ and $1.19\times$. The combination of these techniques leads to a reduced model storage size (76%) with a little decrease in the top-1 prediction accuracy (less than 7%). This work showed that the combination of techniques depends on the architecture of neural network and the reason of optimization: it shows positive impact on the inference time for VGG-16, but it results in a longer inference time for ResNet50 so less benefit in energy consumption for ResNet50 over VGG-16. Tung et al. [78] explored the incorporation of network pruning and weight quantization in a single learning framework named CLIP-Q where both performs in joint and parallel manner. Comparing to the state-of-the-art results, the CLIP-Q technique achieves an improvement in compression rate for AlexNet, GoogLeNet, and ResNet-50 respectively with $51\times$, $10\times$ and $15\times$. Several studies have investigated this compression technique from the hardware perspective. For instance, Faraone et al. [79] suggested a filter pruning framework that utilise efficiently FPGA resources without accuracy

degradation. The evaluation of this approach on the Xilinx KU115 board showed that the pruned AlexNet and TinyYolo networks achieved $2\times$ speedup and $2\times$ reduction in resources (LUTs, DSP, BRAM) without accuracy loss compared to the original networks. Posewsky et al. [80] proposed an FPGA-based accelerator of the pruned DNN inference. This accelerator was implemented on ZedBoard for evaluation. Compared to the software implementation, this approach achieves an improvement with $10\times$ in energy efficiency and $3\times$ in runtime. The hardware implementation of the pruned and non-pruned network achieves an accuracy loss with less than 0.5%. The study by Zhang et al. [81] proposed a compression strategy for CNN based on pruning and quantization and an FPGA-based accelerator for the compressed CNN. The evaluation of the proposed system on Xilinx ZCU104 for AlexNet showed an improvement in terms of latency and throughput on convolutional layers compared with CPU and GPU respectively with $182.3\times$ and $1.1\times$ and an improvement in terms of energy efficiency with $822.0\times$ and $15.8\times$, respectively.

4.3 | Low-rank approximation

Layer decomposition or LRA has been extensively explored to reduce computation complexity to improve efficiency. This method decomposes the model to a compact and approximate one with more lightweight layers by matrix decomposition. Denton et al. [89] applied a LRA of kernels to reduce computation in convolutional layers. The proposed model performed $2.5\times$ speedup with little drop in accuracy (1%). In Ref. [85], Wang et al. proposed a factorised convolutional layer, that outperforms the standard ones on performance and complexity ratio. The factorised network achieved similar performance to VGG-16 while requiring $42\times$ less computation. The authors in Ref. [90] proved that the low-rank approximation technique can also be applied to decompose the weights in the FC layers, which resulted in up to 50% reduction in number of parameters. Following the same strategy, Qui et al. [91] applied LRA on FC layer to reduce the number of weights. With 63% less parameters, this method performed 87.96% accuracy On VGG16-SVD. Also, to decompose pretrained weights, a tucker decomposition is used in Refs. [82,92]. In Ref. [86], LRA was adopted for weights and inputs. Zhang et al. used a generalized Singular Value Decomposition (GSVD) to reduce multiple layers accumulated error. By applying this method on VGG-16, the model achieved $4\times$ speedup with only 0.3% increase in top5 error. Chen et al. proposed a Layer Decomposition-Recomposition Framework (LDRF) [86], in which they applied a Singular Value Decomposition (SVD) to weights matrices. During the SVD decomposition, they lowered the rank of each layer to estimate the layer valid capacity. On VGG-16, the proposed method reached $5.13\times$ speedup with only 0.5% top-5 accuracy reduction. In Ref. [83], the authors showed that low rank tensor decompositions can speedup large CNNs while maintaining performance. The proposed approach achieved $1.82\times$

speedup with $5.0\times$ weights reduction for AlexNet, with low than 0.4% accuracy degradation. The implementation of DNNs can be more effective when using layer decomposition method. Wen et al. designed a new LRA to train a DNN model with lower ranks and higher computation efficiency [87] (Table 4). This method gained $2\times$ speedup on GPU when maintained the accuracy and $4.05\times$ speedup on CPU with low degradation in accuracy. To accelerate the CNN inference computation, Wang et al. proposed an approach based on low rank and group sparse tensor decomposition [88]. On VGG-16, this method achieved $6.6\times$ speedup on CPU with less than 1% degradation on top-5 error. In Ref. [93], the authors proposed a framework to accelerate DNNs based on lower-rank approximation. On FPGA, it achieved an average computation efficiency of 64.5%. LRA can obtain a compact and approximate network model. However, to learn an accurate network structure, LRA needs the reiterations of decomposing, finetuning, etc., resulting in extra computation overhead.

The aim of using the optimization techniques is to reduce model size while maintaining good performance. Lower precision in representing data (quantization) usually improves latency and reduces accuracy especially when dealing with large scale DNNs. Pruning the network also reduces the size of the model and is able to improve accuracy but usually not latency. However, weight quantization is more hardware friendly than weight pruning. LRA techniques are efficient for model compression but the necessity of expensive decompression operations makes it difficult for the implementation. Furthermore, LRA techniques cannot perform global compression of parameters as they are performed layer by layer.

To improve the efficiency and achieve further compression optimization techniques such as pruning and precision reduction or quantization and LRA can be combined.

5 | HW ACCELERATION APPROACHES

DNNs have been successful in a wide range of applications thanks to the rapid development of custom hardware to speed up the training phase as well as the inference phase. Among the different hardware targets previously presented in section 3, FPGA platforms with reconfigurable integrated circuits and embedded hardcodes make it easy to design dedicated hardware accelerators for complex DNNs. In this section, we review many recent research works and summarise acceleration methods based on FPGA.

5.1 | Throughput optimization

Throughput optimization is one of the objectives to design an efficient DNN based accelerator. Several techniques have been explored to achieve higher throughput. However, the most used techniques have included loop optimization, systolic array architecture and Single Instruction Multiple Data (SIMD) based computation.

5.1.1 | Loop optimization

To achieve high throughput, loop optimization techniques such as loop unrolling, loop tiling and loop interchange have been widely used. They have reduced the overheads associated with the massive nested loops which has increased the execution speed. These techniques are based on making effective use of parallel processing capabilities. In Ref. [94], the authors exhaustively analysed loop optimizations and data movement patterns in CNN loops. They provided a new dataflow and architecture, in which they leveraged loop tiling, unrolling, interchange to minimise data communication. Their design achieved 645.25 GOPs of throughput on Intel FPGA using VGG model. Loop tiling is used in Ref. [91] to fit large-scale CNN models into limited on-chip buffers. The proposed approach demonstrated higher acceleration on VGG16-SVD when applying quantization method, and performed 137 GOPs. Also, to explore the design space of dataflow across layers, the authors in Ref. [95] used loop tiling and developed a fused-layer CNN accelerator. The implementation of the proposed approach on a Xilinx FPGA minimised off-chip feature map data transfer by 95% and reached up 61.62 GOPs in throughput. Based on unrolling and tiling loops, Rahman et al. [96] presented ICAN, a 3D compute tile for convolutional layers. With optimization of on-chip buffer sizes for FPGAs, the proposed technique outperformed [95] by 22%. In Ref. [97], loop unrolling is used to define the computation pattern and the data flow. The paper also proposed an RTL compiler ALAMO to automatically integrate the computing primitives to accelerate the operation on FPGA. On AlexNet, the accelerator reported a computational throughput of 114.5 GOPs. In Ref. [98], the authors designed DLAU, an accelerator architecture for large-scale DNNs by exploiting data reuse in order to reduce the memory bandwidth requirements. It included three pipelined processing units to improve the throughput and loop tiling technique to improve locality and minimise data transfer operations. On Xilinx FPGA, the proposed accelerator achieved up to $36.1\times$ speedup with 234mW power consumption.

5.1.2 | Systolic array architecture

Systolic array architecture is another technique that employs high degree of parallelism to improve throughput. It consists of placing, in an organised structure, thousands of Processing Elements (PEs) and connecting them directly to each other to form a large physical matrix of these operators. Each PE has its limited private memory. In Refs. [99–101], systolic array architecture is applied to FPGA-based CNNs. To accelerate CNN/DNN on FPGA, C. Zhang et al. designed and implemented Caffeine [99], a HW/SW co-designed library which decreased underutilised memory bandwidth. The authors proposed a massive number of parallel PEs and organised them as a systolic array to mitigate timing issues for large designs. The implementation of the proposed accelerator on

Xilinx FPGA using VGG performed 636 GOPs. A 1-D systolic array architecture described in OpenCL is proposed in Ref. [101]. This approach is only suitable for small models because all input feature maps are stored in on-chip memory. The implementation of AlexNet on FPGA resulted in 1382 GFLOPS. In this work the DSP utilization is improved by adopting Winograd transformation. In Ref. [100], Wei et al. implemented CNN on Intel FPGA using systolic array architecture which achieved up to 1171 GOPs. In their work they provided an analytical model for resource utilization and performance and developed an automatic design space exploration framework. Besides, the use of current FPGA Computer Aided Design (CAD) tools to synthesise and layout systolic arrays resulted in frequency degradation. In Ref. [102], a 2 D systolic architecture is analysed to identify causes, and two methods are proposed to improve frequency of systolic array designs which is directly related to throughput. The evaluation results attained 1500 GOPs for VGG inference on Xilinx FPGA platform (and achieved $1.29\times$ higher frequency). Table 3 summarises some results.

5.1.3 | SIMD-based computation

To perform high throughput, an SIMD-based computation technique has been used in several recent research works. The authors in Ref. [105] designed a system architecture based on heterogeneous FPGA with DSPs, supporting SIMD paradigm to efficiently process parallel computation for CNNs layers (Convolution and fully connected layers). The proposed architecture required lower computational time (47%) over non-SIMD computational implementation. Furthermore, to accelerate CNNs computation rate on FPGAs, Nguyen et al. proposed Double MAC [103], which is an approach for packing two SIMD MAC operations into a single DSP block with reduced Bitwidth. This work improved the computation throughput by 2 times with the same resource utilization. Zhong et al. designed Synergy [104], a hardware-software co-designed pipelined framework based on heterogeneous FPGA to accelerate CNN inference. Supporting multi-threading, Synergy leveraged all the available on-chip compute resources including CPU, FPGA and NEON SIMD engines. FPGA and the NEON engines are used to accelerate convolutional layers while the CPU cores execute the fully-connected layers and the preprocessing functions. Additionally, a workload accelerator balancing was provided to adopt various networks at runtime without the need to change the hardware or the software implementations. The evaluation of Synergy resulted in higher throughput and energy-efficiency over implementations on the same platform. Likewise, an architecture based on SIMD technique was presented in Ref. [106] to accelerate DNN for speech recognition. SIMD and MIMD (Multiple Instructions Multiple Data) modes are mixed in Ref. [107] to accelerate DL models. In addition, a SIMD like architecture is adopted in Ref. [108] to minimise the energy consumption, which is another important key to further improving accelerator efficiency.

TABLE 3 Pruning effect on DNN models

Pruning technique	Reference	DL Model	Results		
			Without Pruning Top-5 Accuracy	With Pruning Accuracy Loss	Reduction
Weight pruning	[66]	ResNet110	93.50%	0%	90% weight
		ResNet56	93.33%		
	[67]	LeNet-300-100	98.40%	0%	22.9× weight
		LeNet-5	99.20%		71.2× weight
		AlexNet	80.20%		21× weight
	Energy-aware Pruning [68]	AlexNet	80.43%	0.87%	3.7× energy Consumption
		GoogLeNet	88.26%	0.98%	1.6× energy Consumption
Filter pruning	ICLR'17 [69]	VGG-16	-	<1%	34.2% FLOP
		ResNet-110	-		38% FLOP
	[70]	VGG-16	92.77%	0.60%	45% FLOP
		ResNet-18	93.52%	0.30%	24.3% FLOP
		FCN-32s	90.48%	1.30%	55.4% FLOP
		SegNet	86.50%	-2.10%	63.9% FLOP
	Pruned [71]	VGG-16 on CIFAR10	93.23%	-0.78%	90.50% FLOP
		VGG-16 on GTSRB	99.31%	0.55%	96.6% FLOP
Channel pruning	DCP [72]	ResNet-50	92.93%	-0.14%	30% channels
	[73]	VGG-16	-	<1%	5.64× FLOP
		ResNet-110	-	0.08%	2.48× FLOP
	[74]	VGGNet on CIFAR-10	93.66%	-0.14%	51.0% FLOP
		VGGNet on CIFAR-100	73.26%	-0.22%	37.1% FLOP
		VGGNet on SVHN	97.83%	-0.11%	50.1% FLOP

5.2 | Energy optimization

Reducing the energy consumption is a key challenge for designing an efficient DNN-based accelerator. Therefore, various techniques have been explored by researchers to obtain high throughput with low energy consumption.

5.2.1 | Reducing the memory bandwidth

Many recent researchers focussed on reducing the on-chip and off-chip memory bandwidth. In [109] Zhang et al., presented a 2-D interconnection between PEs and local memory to minimise the on-chip memory bandwidth. The authors also, increased the data locality, which reduced the off-chip memory requirements. Using OpenCL, the design implementation of VGG on FPGA achieved a 1790 GOPs throughput and an energy efficiency of 47.78 GOPs/W. Also, Memsqueezer [110], an on-chip memory subsystem for low-overhead DL

accelerator, that can be implemented on FPGA is proposed. It compressed the data and weights from the hardware perspective and eliminate the data redundancy. With Memsqueezer buffers, CNN accelerators achieve 80% energy reduction over conventional buffer designs with the same area budget. Reducing data transfer between on-chip memory and off-chip memory can also minimise the energy consumption. It is in this context, Shen et al. [111] realized a CNN accelerator with a flexible data buffering scheme Escher. This latter reduced the bandwidth requirements by 2.4× on FPGA using AlexNet. The study by Li et al. [112] observed that for CNN accelerators, over 80% of the energy are consumed by DRAM accesses. The authors proposed SmartShuttle, an adaptive layer, to minimise the off-chip memory accesses by investigating the impact of sparsity and reusability of data on the memory. The evaluation on AlexNet showed that SmartShuttle reduced up to 47.6% DRAM access volume, and reached up 36% of energy savings. In the same context, Ref. [113] designed an algorithm called block convolution to completely

TABLE 4 Low-rank approximation effect on DNN models

Reference	DL Model	Results			
		Without LRA Top-5 Accuracy	With LRA		
			Accuracy Loss	Reduction	Speed up On CPU
[82]	AlexNet	80.03%	1.70%	5.46× weight 2.67× FLOP	2.72×
	VGG-S	84.60%	0.55%	7.40× weight 4.80× FLOP	3.68×
	GoogLeNet	88.90%	0.24%	1.28× weight 2.06× FLOP	1.42×
	VGG-16	89.90%	0.50%	1.09× weight 4.93× FLOP	3.34×
[83]	AlexNet	80.03%	0.34%	5.00× weight	1.82×
	VGG-16	90.60%	0.29%	2.75× weight	2.05×
	GoogLeNet	92.21%	0.42%	2.84× weight	1.20×
[84]	VGG-16	89.90%	0.30%	-	3.80×
[85]	VGG-16	90.10%	0%	22× params	14×
				42× FLOP	
[86]	VGG-16	89.90%	0.50%	-	5.13×
[87]	AlexNet in Caffe	80.03%	1.71%	-	4.05×
[88]	VGG-16	-	<1%	-	6.6×

avoid the off-chip intermediate data transfers. It performed high throughput on FPGA using VGG-16. The fused-layer CNN accelerator proposed by Alwani et al. [95] also minimised the off-chip data transfer by 95%.

5.2.2 | Other approaches

Many other methods have been used to reduce power consumption. In Ref. [114], Zhang et al. proposed a deeply pipelined FPGA architecture to leverage the design space for energy efficiency. The evaluation results on VGG-16 achieved 8.28 GOPs/J of energy efficiency. The study by Zhu et al. [115] showed that using low-rank approximation, a 31% to 53% energy reduction can be reached. The low data representation can also reduce energy consumption, the binarised neural networks in Ref. [116] attained 44.2 GOPs/W. In Ref. [61], Han et al. presented an Efficient Speech Recognition Engine (ESE), a SW/HW co-design framework which works directly on compressed LSTM model. It achieved up to 428 FPs/W which is 40× more energy efficient than CPU. Li et al. [117] presented the Efficient RNN (E-RNN) framework for FPGA implementation of the Automatic Speech Recognition (ASR) application. For more accurate block-circulant training, they used the Alternating Direction Method of Multipliers (ADMM) technique. This approach achieves 37.4× energy efficiency improvement respectively compared with ESE [61].

Table 4 illustrates several accelerations techniques that optimised low energy consumption.

5.3 | Algorithmic optimization

Recent works [59,118,119] have demonstrated that applying mathematical optimizations such as Fast Fourier (FF) and Winograd algorithms to DNNs accelerators can improve resource productivity and efficiency. These transformations can decrease the required MAC operations in the layer's network by reducing the model arithmetic complexity. For example, each element in the output feature map of a CNN model is computed individually. Contrariwise, FF and Winograd algorithms transform the input feature map and filter to corresponding domain (Winograd or frequency) and then perform element-wise matrix multiplication [120]. To get the final output an inverse transformation is applied. The reduction of the model arithmetic complexity depends on the parameters of the algorithm. With 8×8 input tile size; FF Transform (FFT) algorithm can reduce the multiplication by 3.45 times for 3×3 filters. On the other hand, with 6×6 input tile size, Winograd algorithm can reduce the multiplication by 4 times for 3×3 filters. In Ref. [120], Liang et al. investigated both Winograd and Fast Fourier transformations and proved their considerable effect in reducing arithmetic complexity and improving CNNs performance on FPGAs.

5.3.1 | Fast Fourier transform

FFT is a well-known approach that reduces the computational complexity. The study by Lin et al. [118] presented a framework based on FFT that achieved significant processing speed and reduction in storage requirement. Zhang et al, exploited FFT also to deal with the complexity of the convolutional layers computation [119]. The proposed design performed 123.48 GFLOPs on Intel Quick-Assist QPI FPGA Platform using VGG. To accelerate operations in each convolutional layer too, a tile-based FFT algorithm (tFFT) is presented in Ref. [121]. Another proposed framework, C-LSTM [59], used FFT to accelerate the LSTM inference by reducing the computational and storage complexities. The latter performed $18.8\times$ and $33.5\times$ gains for performance and energy efficiency compared with the state-of-the-art ESE [61], respectively.

5.3.2 | Winograd algorithm

Very similar to FFT, Winograd, the fast matrix multiplication algorithm is applied to DNNs to minimise the multiplication requirement. By adopting Winograd transformation in Ref. [101], the DSP utilization is improved. Lu et al. [120] used Winograd algorithm to accelerate CNNs by reducing the multiplication operations and saving DSP resources. On VGG, the proposed design attained 2479.6 GOPs of throughput. Additionally, the study by Di Cecco et al. [122] implemented a Winograd convolution engine on FPGA which performed 55 GOPs when executing VGG. More recently, Huang et al. [123] designed an accelerator based on Winograd algorithm. In this work, the authors evaluated Winograd algorithm with different tile sizes. When using VGG, the design achieved 943 GOPs on FPGA. More details are presented in Table 5.

5.4 | HW design automation

Design automation frameworks have been explored to accelerate DNNs by automatically map their models onto hardware platforms. The use of such frameworks can significantly simplify the development and speedup the automatic generation of the hardware accelerator. Some approaches have focussed on using the HLS which is an automated design process that generates high-performance FPGA hardware from software. The study by Zhang et al. [99] designed Caffeine, a HW/SW co-designed library based on HLS tools. Kim et al. [124] analysed the efficiency of the HLS implementation and designed a CNN based FPGA accelerator using LegUp HLS tool. The proposed accelerator performed 138 GOPs on VGG-16. SDAccel, OpenCL, HLS tools are applied in Ref. [122] to synthesise a CNN accelerator that reached 55 GFLOPS on VGG. In Ref. [63], Zhang et al. applied HLS for the implementation of Long-term Recurrent Convolution Network (LRCN) on Xilinx FPGA based on their designed resource allocation scheme REALM. More recently, the authors in Ref. [125] presented an implementation of Neural Machine Translation (NMT) model on FPGA. It used HLS to build parameterised IPs. Many other approaches have used Register Transfer Level (RTL) which describes the design as the transfers that occur between registers every clock cycle. RTL leveraging offers higher performance. In Ref. [97], Ma et al. proposed an RTL-level CNN compiler that automatically generates customised FPGA accelerator. The VGG implementation gained $2.7\times$ throughput improvement over [99]. The study by Ma et al. [126] developed an RTL FPGA-based accelerator which achieved 720.15 GOPs using VGG-16. Using RTL codes, the designed accelerator in Ref. [127] achieved 638.9 GOPs on VGG-16. The study by Zeng et al. [128] used RTL IPs to

TABLE 5 FPGA-based CNN accelerators optimising throughput

Acceleration technique	DL Model	Design Tool	DSP Utilization	Throughput (GOPs)	Reference
Loop optimization	VGG-16	Verilog/ Quartus Prime	1518	645.25	[94]
	VGG16-SVD	HDL	780	136.97	[91]
	AlexNet	C++/HLS	2401	61.62	Fused-layer [95]
	AlexNet	Verilog	2594	75.16	ICAN [96]
	AlexNet	RTL	256	114.5	ALAMO [97]
	DNN (MNIST)	-	167	-	DLAU [98]
systolic-Like architecture	VGG-16	HLS	2833	636	Caffeine [99]
	VGG-16	C/C++/HLS	1500	1171.3	[100]
	AlexNet	OpenCL	1476	1382	DLA [101]
	VGG-16	C/HLS	1368	1495	[102]
SIMD	VGG-16	Verilog RTL	2240	-	Double MAC [103]
	CNN (MNIST)	C/HLS	-	2.15 (96.2 frames/s)	Synergy [104]
	VGG-16	-	880	425.32	[105]

TABLE 6 FPGA-based CNN accelerators optimising low energy consumption

Acceleration technique	DL Model	Design	BRAM Utilization	DSP Utilization	Energy Efficiency	Throughput	Reference
-2-D multi-cast interconnection	VGG-16	OpenCL	1450/2713	2756	47.78	1790 GOPs	[109]
Between PEs and local memory							
-Increase the data locality							
Memory subsystem	CNN	RTL	-	-	80%	2×	Memsqueezer [110]
Optimising off-chip memory	AlexNet	Verilog/ Synopsys	47.6% DRAM Access reduction	-	36%	-	SmartShuttle [112]
Avoiding off-chip data transfers:	VGG-16	-	1090	900	-	374.98 GOPs	Block convolution [113]
Multi-layer fusion							
loop tiling							
Pipelined FPGA cluster	VGG-16	HDL	-	-	8.28 (GOPs/J)	290 GOPs	[114]
Flexible data buffering scheme	AlexNet	RTL/HLS	1745 (59%)	2182	2.4×peak bandwidth Reduction	135 GOPs	Escher [111]
Low-rank approximation	DNN (SVHN)	Verilog RTL/ Synopsys	-	-	31% to 53% energy Reduction	22% to 43% Throughput increase	LRADNN [115]
Binarised neural networks	CNN	C++/HLS	86-94/140	3/220	44.2 GOPs/W	207.8 GOPs	BNN [116]
Compression (quantization + pruning)	LSTM	-	1080	1504	428 FPs/W	282 GOPs	ESE [61]
HW/SW codesigned framework							

create a reconfigurable framework for deploying CNN-RNN models on FPGAs. On LRCN network, the designed hardware system performed up to 690.76 GOPs throughput and achieved 86.34 GOPs/W energy efficiency. More results are provided in Table 6. Some other approaches combined the finer level optimization of RTL and the flexibility of HLS to design DNNs accelerators which achieved 114.5 GOPs in Ref. [129]. Based on RTL-HLS hybrid library, Guan et al. designed FP-DNN [130], a framework to automatically generate optimised DNNs implementations on FPGA. The evaluation results reached 364.36 GOPs on CNN model and 315.85 GOPs on RNN model.

The acceleration methods aim to speed up DNNs while improving throughput and reducing energy consumption. Several techniques have been explored to achieve higher throughput such as loop optimization, systolic array architecture and SIMD based computation. A DNN accelerator designed using these techniques usually consumes higher energy. Therefore, various techniques have been explored to obtain high throughput with low energy consumption such as

memory bandwidth reduction and model compression. For further improvement, algorithmic optimization approaches like Fast Fourier and Winograd algorithms can be used. Furthermore, the automatic generation of high-performance hardware accelerator from software can significantly simplify the development and speed up the process (e.g. HLS). Reducing the energy consumption and improving throughput are key challenges for designing an efficient DNN-based accelerator. Therefore, various acceleration techniques can be combined along with the optimization approaches.

6 | CONCLUSION

Herein, DL concept was initially presented through the complexity of different models. We also reviewed the exploration of different computation platforms of DL implementation. Then, we discussed a review of the literature about the different approaches used to optimise DL models to make them more hardware friendly. In the end, we presented and analysed the

TABLE 7 DNN accelerators employing computational transform

Algorithm	DL Model	Design	DSP Utilization	Throughput	Energy Efficiency	Reference
FFT	VGG16	-	224	123.48 GOPs	9.37 GOPs/W	[119]
	Google LSTM	C/C++/HLS	2786	330 275 FPs	14 359 FPs/W	C-LSTM [59]
	Small LSTM		2347	559,257 FPs	25,420 FPs/W	C-LSTM [59]
Winograd	VGG	C/HLS	2520	2940.7 GOPs	124.6 GOPs/W	[120]
	VGG	OpenCL/HLS	1307	55 GOPs	-	[122]
	VGG	C/HLS	756	943 GOPs	74.5× (over CPU)	[123]

TABLE 8 DNN accelerators employing design automation

Design	DL Model	DSP Utilization	Throughput (GOPs)	Energy Efficiency	Reference
HLS	VGG-16	2833	354	-	Caffeine [99]
	VGG-16	380	138	41.8 GOPs/W	[124]
	VGG-16	1307	55	-	[122]
	NMT	5969	14.8	-	[125]
RTL	VGG-16	3600	720.15	-	[126]
	VGG-16	2967	638.9	-	CaFPGA [127]
	LRCN	1248	690.76	86.34 GOPs/W	[128]
RTL-HLS	AlexNet	256	114.5	-	[129]
	VGG-19	1036	364.36	14.57 GOPs/J	FP-DNN [130]
	LSTM-LM	1036	315.85	12.63 GOPs/J	FP-DNN [130]

used acceleration techniques for the deployment of DL models on FPGA platforms. The deployment of DL on embedded equipments with high accuracy, high throughput and low consumption is still a challenge. Indeed, hardware constraints required for lower power consumption, such as limited processing power, lower memory footprint, and less bandwidth, reduce the accuracy. Due to the increasing complexity of DNN models it is difficult to integrate a large DNN into an embedded hardware design. This made researches think about applying optimization and acceleration techniques. Optimization techniques focussed on modifying DL algorithms to make them more hardware-friendly. They effectively digest the redundancy of models and provide improved computing efficiency with minimal loss of accuracy. However, the acceleration methods aim to speed up DNNs while improving throughput and reducing energy consumption. Also, applying algorithmic optimization like Fast Fourier and Winograd algorithms can accelerate DNNs and improve resource productivity and efficiency. In addition, the use of frameworks to automatically map models onto hardware platforms simplifies the development and speedup the automatic generation of the hardware acceleration. The efficient implementation of complex DNN models on new and increasingly powerful embedded platforms can offer many benefits for AI applications. Previous works faced challenges such as limited hardware resources, long development time, and performance degradation. Moreover, it is difficult to use all the functionalities of neural network algorithms in hardware compared to software implementations [131]. In this context,

new FPGAs, using parallel processing and embedded programmable cores have advantages over other hardware platforms for DNN implementations. Whole systems can be integrated on a chip using many hardware components such as memories, fast devices, DSP units and processor cores which expedite the design of such large-scale systems. FPGAs are very flexible and allow reconfiguration to optimise bit resolution, clock rate, parallelisation, and pipeline processing for a given application. Some FPGA manufacturers like Xilinx have provided accelerators (DPU) along with other tools and APIs to optimise pre-trained DL models by applying pruning and quantization techniques.

ORCID

Meriam Dhouibi  <https://orcid.org/0000-0002-0273-3262>

REFERENCES

- Li, Y. et al.: Face recognition based on recurrent regression neural network. *Neurocomputing*. 297, 50–58 (2018)
- Marra, F. et al.: A deep learning approach for iris sensor model identification. *Pattern Recogn. Lett.* 113, 46–53 (2018)
- Lee, J.G., et al.: Deep learning in medical imaging: general overview. *Korean J. Radiol.* 18(4), 570–584 (2017)
- Justesen, N. et al.: Deep learning for video Game Playing. *IEEE Trans. Games* (2019)
- Lecun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature*. 521, 436–444 (2015)
- Fan, K., Wen, S., Deng, Z.: Deep learning for detecting Breast cancer Metastases on WSI. In: *Innovation in Medicine and Healthcare Systems, and Multimedia*, pp. 137–145. Springer, Singapore (2019)

7. Wang, J. et al.: Deep learning for smart manufacturing: methods and applications. *J. Manuf. Syst.* 48, 144–156 (2018)
8. Rémy, S.: Apprentissage profond et acquisition de représentations latentes de séquences peptidiques'. <https://hal.inria.fr/hal-01406368> (2016). Accessed 25 March 2018
9. Groumpos, P.P.: Deep learning vs. Wise learning: a critical and challenging overview. *IFAC-PapersOnLine*. 49(29), 180–189 (2016)
10. Ackerman, E.: How drive. AI is mastering autonomous driving with deep learning, *IEEE Spectrum*, 1. <https://spectrum.ieee.org/cars-that-think/transportation/self-driving/how-driveai-is-mastering-autonomous-driving-with-deep-learning> (2017). Accessed 20 March 2019
11. Giusti, A., et al.: A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robot. Autom. Lett.* (2016)
12. Esteva, A., et al.: Dermatologist-level classification of skin cancer with deep neural networks. *Nature*. 542(7639), 115 (2017)
13. Khagi, B., Lee, C.G., Kwon, G.R.: Alzheimer's disease Classification from Brain MRI based on transfer learning from CNN. In: *BMEiCON 2018 - 11th Biomedical Engineering International Conference*, Chiang Mai, 21–24 November 2018 (2019)
14. Gilmer, J., et al.: Neural message passing for quantum chemistry. In: *Proceedings of the 34th International conference on machine learning*, Vol. 70, *JMLR. org*, Sydney, NSW, pp. 1263–1272 (2017)
15. Zhang, J., et al.: Deep neural networks in machine translation: an overview. *IEEE Intelligent Systems*. 30(5), 16–25 (2015)
16. Owens, A., et al.: Visually indicated sounds. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2405–2413 (2016)
17. Karpathy, A., Fei, L.: Deep Visual-Semantic Alignments for generating image descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.* 39(4), 664–676 (2017)
18. He, K. et al.: Deep residual learning for image recognition. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, 27–30 June 2016 (2016)
19. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: *3rd International conference on learning representations, ICLR 2015 - Conference Track Proceedings*, San Diego, CA, 7–9 May 2015 (2015)
20. Canziani, A., Culurciello, E., Paszke, A.: Analysis of deep neural network architectures for practical applications. *CoRR*. abs/1605.07678. <http://arxiv.org/abs/1605.07678> (2016)
21. Horowitz, M.: 1.1 computing's energy problem (and what we can do about it). In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, San Francisco, CA, 9–13 February 2014, IEEE, pp. 10–14 (2014)
22. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*, Lake Tahoe, Nevada, December 2012, pp. 1097–1105 (2012)
23. Szegedy, C., et al.: Going deeper with convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, 8–10 June 2015, pp. 1–9 (2015)
24. Sze, V. et al.: Hardware for machine learning: challenges and opportunities. In: *2017 IEEE Custom Integrated Circuits Conference (CICC)*, Austin, TX, 30 April–3 May 2017, pp. 1–8, IEEE (2017)
25. Oliveira, D., et al.: Experimental and analytical study of xeon phi reliability. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, p. 28. (2017)
26. Frank, B.H.: Google's new chip makes machine learning way faster. <https://www.computerworld.com/article/3072652/googles-new-chip-makes-machine-learning-way-faster.html> (2016). Accessed 6 May 2018
27. Conformant products - the Khronos group Inc. (2019). <https://www.khronos.org/conformance/adopters/conformant-products/#|opencl>
28. NVDLA Primer — NVDLA Documentation. <http://nvdla.org/primer.html> (2020). Accessed 7 May 2020
29. Tayal, P.: AMD's new Vega GPUs target deep learning. <https://marketrealist.com/2018/12/amds-new-vega-gpus-target-deep-learning/> (2018). Accessed 15 April 2020
30. Bacon, D.F., et al.: FPGA programming for the masses. *Commun. ACM*. 56(4), 56–63 (2013)
31. Lacey, G., Taylor, G.W., Areibi, S.: Deep learning on fpgas: Past, present, and future. *arXiv preprint arXiv:160204283* (2016)
32. Hou, X. et al.: Vehicle licence plate recognition system based on deep learning deployed to PYNQ. In: *Iscit 2018 - 18th International Symposium on Communication and Information Technology*, Bangkok, 26–29 September 2018(2018)
33. Ranjan, R., Patel, V.M., Chellappa, R.: HyperFace: a deep multi-task learning framework for Face detection, Landmark Localization, Pose estimation, and Gender recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 41, pp. 121–135 (2019)
34. Gaide, B. et al.: Xilinx adaptive compute acceleration platform: Versal™; architecture. In: *Fpga 2019 - Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Seaside, CA, February 2019 (2019)
35. XILINX. DPU: For convolutional neural network - DPU IP product Guide. <https://www.xilinx.com/products/intellectual-property/dpu.html> (2019). Accessed 17 April 2020
36. Alveo. <https://www.xilinx.com/products/boards-and-kits/alveo.html> (2020). Accessed 7 May 2020
37. MPC-X Series | Maxeler Technologies. <https://www.maxeler.com/products/mpc-xseries/> (2020). Accessed 9 May 2020
38. Feldman, M.: Microsoft goes all in for FPGAs to build out AI cloud | TOP500 Supercomputer Sites. <https://www.top500.org/news/microsoft-goes-all-in-for-fpgas-to-build-out-cloud-based-ai/> (2017). Accessed 9 May 2020
39. Amazon: Amazon EC2 F1 instances (2019). <https://aws.amazon.com/ec2/instance-types/f1/> Accessed 9 May 2020
40. FPGA cloud server. <https://cn.aliyun.com/product/ecs/fpga> (2019). Accessed 9 May 2020
41. FPGA accelerated cloud server. <https://www.huaweicloud.com/product/fcs.html> (2019). Accessed 9 May 2020
42. FPGA cloud server_FPGA instance_hardware acceleration-Tencent Cloud. <https://cloud.tencent.com/product/fpga> (2019). Accessed 10 May 2020
43. FPGA cloud server_Baidu Cloud. <https://cloud.baidu.com/product/fpga.html> (2019). Accessed 10 May 2020
44. Chen, Y.H. et al.: Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid State Circ.* (2017)
45. Cloud, TPU. <https://cloud.google.com/tpu> (2020)
46. Linley, G.: Habana Wins Cigar for AI inference: startup takes performance lead with Mystery architecture. <https://www.linleygroup.com/mpc/article.php?id=12103> (2019). Accessed 4 May 2020
47. Lightspeur 2801 neural accelerator for edge devices. <https://www.gyrfalcontech.ai/solutions/2801s/> (2019). Accessed 10 May 2020
48. Synced, California: Startup GTI Releases AI chips to challenge NVIDIA and Intel (2019). <https://syncedreview.com/2019/01/28/california-startup-gti-releases-ai-chips-to-challenge-nvidia-and-intel/>
49. Sheng, T. et al.: A quantization-friendly separable convolution for MobileNets. In: *Proceedings - 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications*, Williamsburg, VA, 25–25 March 2018, EMC2 2018 (2018)
50. Zhou, A. et al.: Incremental network quantization: towards lossless cnns with low-precision weights. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, Toulon, France, 24–26 April 2017 (2019)
51. Li, F., Zhang, B., Liu, B.: Ternary weight networks. *arXiv preprint arXiv:160504711* (2016)
52. Rastegari, M. et al.: XNOR-net: Imagenet classification using binary convolutional neural networks. In: *Lecture Notes in Computer Science (including subseries Lecture notes in Artificial Intelligence and Lecture notes in Bioinformatics)*. Springer, Cham, Amsterdam, The Netherlands (2016)
53. Gysel, P.: Ristretto: hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:160506402* (2016)

54. Zhou, S.C. et al.: Balanced quantization: an effective and efficient approach to quantized neural networks. *J. Comput. Sci. Technol.* 32(4), 667–682 (2017)
55. Hubara, I. et al.: Quantized neural networks: training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18(1), 6869–6898 (2017)
56. Cai, Z. et al.: Deep learning with low precision by half-wave Gaussian quantization. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5918–5926 (2017)
57. Courbariaux, M. et al.: Binarised neural networks: training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830* (2016)
58. Zhou, S. et al.: Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.01660* (2016)
59. Wang, S., et al.: 'C: Enabling efficient LSTM using structured compression techniques on FPGAs. In: *FPGA 2018 - Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2018 (2018)
60. Moss, D.J.M., et al.: A customisable matrix multiplication framework for the intel harp2 xeon+ fpga platform: a deep learning case study. In: *Proceedings of the 2018 ACM/SIGDA International Symposium on field-Programmable Gate Arrays*, Monterey, CA, February 2018, ACM, pp. 107–116 (2018)
61. Han, S., et al.: ESE: efficient speech recognition engine with sparse lstm on fpga. In: *Proceedings of the 2017 ACM/SIGDA International Symposium on field-Programmable Gate Arrays*, ACM, pp. 75–84 (2017)
62. Shen, J. et al.: Towards a uniform template-based architecture for accelerating 2d and 3d cnns on fpga. In: *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2018, ACM, pp. 97–106 (2018)
63. Zhang, X., et al.: High-performance video content recognition with long-term recurrent convolutional network for FPGA. In: *27th International Conference on field programmable logic and applications (FPL)*, Ghent, 4–8 September 2017, IEEE, pp. 1–4 (2017)
64. Hu, Q., Wang, P., Cheng, J.: From hashing to cnns: training binary weight networks via hashing. In: *Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, 2–7 February 2017 (2018)
65. Wang, P., Cheng, J.: Fixed-point factorised networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, 21–26 July 2017, pp. 4012–4020 (2017)
66. Carreira-Perpinán, M.A., Idelbayev, Y.: "Learning-Compression" algorithms for neural net pruning. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, 18–23 June 2018, pp. 8532–8541 (2018)
67. Zhang, T., et al.: A systematic DNN weight pruning framework using alternating direction method of multipliers. In: *Lecture Notes in Computer Science (including subseries Lecture notes in Artificial Intelligence and Lecture notes in Bioinformatics)*. Springer, Cham, Munich, Germany (2018)
68. Yang, T.J., Chen, Y.H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. In: *Proceedings of the IEEE Conference on computer vision and pattern recognition*, Honolulu, HI, 21–26 July 2017, pp. 5687–5695 (2017)
69. Li, H. et al.: Pruning filters for efficient convnets. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, Toulon, France, 24–26 April 2017 (2019)
70. Huang, Q. et al.: Learning to prune filters in convolutional neural networks. In: *IEEE Winter Conference on Applications of Computer Vision (WACV)*, Lake Tahoe, NV, 12–15 March 2018, pp. 709–718, IEEE (2018)
71. Singh, P. et al.: Multi-layer pruning framework for compressing single shot multibox detector. In: *IEEE Winter Conference on Applications of Computer Vision (WACV)*, Waikoloa Village, HI, 7–11 January 2019, IEEE, pp. 1318–1327 (2019)
72. Zhuang, Z., et al.: Discrimination-aware Channel pruning for deep neural networks. In: *Advances in Neural Information Processing Systems*. Curran Associates Inc., Montreal, Canada (2018)
73. Liu, C., Wu, H.: Channel pruning based on mean gradient for accelerating convolutional neural networks. *Signal Process.* 156, 84–91 (2019)
74. Liu, Z., et al.: Learning efficient convolutional networks through network slimming. In: *Proceedings of the IEEE International Conference on Computer Vision*, Venice, 22–29 October 2017 (2017)
75. Sze, V., et al.: Efficient processing of deep neural networks. *Synthesis Lectures on Computer Architecture*. 15(2), 1–341 (2020)
76. He, Y., et al.: AutoML for model compression and acceleration on mobile devices. In: *Lecture notes in computer science (including subseries Lecture notes in Artificial Intelligence and Lecture notes in Bioinformatics)*. Springer, Cham, Munich, Germany (2018)
77. Qin, Q., et al.: To compress, or not to compress: Characterising deep learning model compression for embedded inference. In: *Proceedings - 16th IEEE International Symposium on Parallel and Distributed Processing with Applications*, 17th IEEE International Conference on Ubiquitous Computing and Communications, 8th IEEE International Conference on Big Data and Cloud Computing, 11th IEEE International Conference on Social Computing and Networking and 8th IEEE International Conference on Sustainable Computing and Communications, Melbourne, Australia, 11–13 December 2018. ISPA/IUCC/BDCloud/SocialCom/SustainCom 2018. (2019)
78. Tung, F., Mori, G.: CLIP-Q: Deep network compression learning by in-parallel pruning-quantization. In: *Proceedings of the IEEE computer Society Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, 18–23 June 2018 (2018)
79. Faraone, J. et al.: Customising low-precision deep neural networks for FPGAs. In: *28th International Conference on Field Programmable Logic and Applications (FPL)*, Dublin, 27–31 August 2018, pp. 97–973, IEEE (2018)
80. Posewsky, T., Ziener, D.: A flexible FPGA-based inference architecture for pruned deep neural networks. In: *Lecture Notes in Computer Science (including subseries Lecture notes in Artificial Intelligence and Lecture notes in Bioinformatics)*. Springer, Cham, Braunschweig, Germany (2018)
81. Zhang, M. et al.: Optimised compression for implementing convolutional neural networks on FPGA. *Electronics (Switzerland)* (2019)
82. Kim, Y.D. et al.: Compression of deep convolutional neural networks for fast and low power mobile applications. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, San Juan, Puerto Rico, 2–4 May 2016 (2016)
83. Tai, C. et al.: Convolutional neural networks with low-rank regularisation. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, San Juan, Puerto Rico, 2–4 May 2016 (2016)
84. Zhang, X. et al.: Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 38(10), 1943–1955 (2015)
85. Wang, M., Liu, B., Foroosh, H.: Factorized convolutional neural networks. In: *Proceedings - 2017 IEEE International Conference on computer vision Workshops*, Venice, 22–29 Oct. 2017. ICCVW (2017)
86. Chen, W. et al.: A layer decomposition-recomposition framework for neuron pruning towards accurate lightweight networks. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, Honolulu, Hawaii, January 27–February 1, 2019 (2019)
87. Wen, W. et al.: Coordinating filters for faster deep neural networks. In: *Proceedings of the IEEE International Conference on Computer Vision*, Venice, 22–29 October 2017 (2017)
88. Wang, P., Cheng, J.: Accelerating convolutional neural networks for mobile applications. In: *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, Amsterdam, The Netherlands, October 2016 (2016)
89. Denton, E. et al.: Exploiting linear structure within convolutional networks for efficient evaluation. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)*. MIT Press, Cambridge, MA, Vol. 1, 1269–1277 (2014)
90. Sainath, T.N. et al.: Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets. In: *ICASSP*,

- IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, Vancouver, BC, Canada, 26-31 May 2013 (2013)
91. Qiu, J., et al.: Going deeper with embedded FPGA platform for convolutional neural network. In: *Fpga 2016 - Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2016 (2016)
92. Ding, H., et al.: A compact CNN-DBLSTM based character model for offline handwriting recognition with tucker decomposition. In: *Proceedings of the International Conference on Document Analysis and Recognition*, Kyoto, 9-15 November 2017. ICDAR (2017)
93. Li, B. et al.: Running sparse and low-precision neural network: when algorithm meets hardware. In: *Proceedings of the Asia and South Pacific Design Automation Conference*, Jeju, 22-25 January 2018. ASP-DAC (2018)
94. Ma, Y. et al.: Optimising loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In: *Fpga 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on field-programmable gate arrays*, Monterey, CA, February 2017 (2017)
95. Alwani, M. et al.: Fused-layer CNN accelerators. In: *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, Taipei, Taiwan, 15-19 October 2016 (2016)
96. Rahman, A., Lee, J., Choi, K.: Efficient FPGA acceleration of Convolutional Neural Networks using logical-3D compute array. In: *Proceedings of the 2016 Design, Automation and Test in Europe Conference and Exhibition, DATE 2016*, Dresden, Germany, 14-18 March 2016 (2016)
97. Ma, Y. et al.: FPGA acceleration of deep learning algorithms with a modularised RTL compiler. *Integration*. 62, 14–23 (2018)
98. Wang, C. et al.: DLAU: a scalable deep learning accelerator unit on FPGA. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* 36, 513–517 (2017)
99. Zhang, C. et al.: Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* 38, 2072–2085 (2019)
100. Wei, X., et al.: Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In: *Proceedings - Design Automation Conference*, Austin, TX, 18-22 June 2017 (2017)
101. Aydonat, U. et al.: An OpenCL™ deep learning accelerator on Arria 10. In: *Fpga 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2017 (2017)
102. Zhang, J. et al.: Frequency improvement of systolic array-based CNNs on FPGAs. In: *Proceedings - IEEE International Symposium on Circuits and Systems*, Sapporo, Japan, 26-29 May 2019 (2019)
103. Nguyen, D., Kim, D., Lee, J.: Double-MAC: Doubling the performance of convolutional neural networks on modern FPGAs. In: *Proceedings of the 2017 design, automation and test in Europe, DATE 2017*, Lausanne, 27-31 March 2017 (2017)
104. Zhong, G. et al.: Synergy: an HW/SW framework for high throughput CNNs on embedded heterogeneous SoC. *ACM Trans. Embed. Comput. Syst.* (2019)
105. Spagnolo, F. et al.: Energy-efficient architecture for CNNs inference on heterogeneous FPGA. *J. Low. Power Electron. Appl.* (2020)
106. Price, M., Glass, J., Chandrakasan, A.P.: A scalable speech recogniser with deep-neural-network acoustic models and voice-activated power gating. In: *Digest of Technical Papers - IEEE International Solid-State Circuits Conference* (2017)
107. Yazdambakhsh, A. et al.: A unified MIMD-SIMD acceleration for generative adversarial networks. In: *Proceedings - International Symposium on computer architecture*, Los Angeles, CA, June 2018 (2018)
108. Lin, C.Y., Lai, B.C.: Supporting compressed-sparse activations and weights on SIMD-like accelerator for sparse convolutional neural networks. In: *Proceedings of the Asia and South Pacific Design Automation Conference*. ASP-DAC. (2018)
109. Zhang, J., Li, J.: Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network. In: *Fpga 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2017 (2017)
110. Wang, Y., Li, H., Li, X.: Re-architecting the on-chip memory sub-system of machine-learning accelerator for embedded devices. In: *IEEE/ACM International Conference on Computer-Aided Design*, Digest of Technical Papers, Austin, TX, 7-10 November 2016, ICCAD (2016)
111. Shen, Y., Ferdman, M., Milder, P.: Escher: A CNN accelerator with flexible buffering to minimise off-chip transfer. In: *Proceedings - IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, 30 April-2 May 2017, FCCM (2017)
112. Li, J., et al.: SmartShuttle: optimising off-chip memory accesses for deep learning accelerators. In: *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition*, Dresden, 19-23 March 2018. DATE 2018 (2018)
113. Li, G. et al.: Block convolution: towards memory-efficient inference of large-scale CNNs on FPGA. In: *Proceedings of the 2018 design, automation and test in Europe Conference and Exhibition, DATE 2018*, Dresden, 19-23 March 2018 (2018)
114. Zhang, C. et al.: Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. In: *Proceedings of the International Symposium on Low Power Electronics and Design*, San Francisco Airport, CA, August 2016 (2016)
115. Zhu, J., Qian, Z., Tsui, C.Y.: LRADNN: High-throughput and energy-efficient deep neural network accelerator using low rank approximation. In: *Proceedings of the Asia and South Pacific Design Automation Conference*, Macao, China, 25-28 January 2016. ASP-DAC. (2016)
116. Zhao, R., et al.: Accelerating binarised convolutional neural networks with software-programmable FPGAs. In: *Fpga 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2017 (2017)
117. Li, Z., et al.: E-RNN: Design optimization for efficient recurrent neural networks in FPGAs'2019. In: *Proceedings - 25th IEEE International Symposium on High Performance Computer Architecture*, Washington, DC, 16-20 February 2019. HPCA (2019)
118. Lin, S., et al.: FFT-based deep learning deployment in embedded systems. In: *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition*, Dresden, 19-23 March 2018. DATE 2018 (2018)
119. Zhang, C., Prasanna, V.: Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system. In: *Fpga 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, February 2017 (2017)
120. Lu, L. et al.: Evaluating fast algorithms for convolutional neural networks on FPGAs. In: *Proceedings - IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, 30 April-2 May 2017. FCCM 2017 (2017)
121. Lin, J., Yao, Y.: A fast algorithm for convolutional neural networks using tile-based fast Fourier transforms. *Neural Process. Lett.* (2019)
122. Di Cecco, R. et al.: FPGA framework for convolutional neural networks. In: *Proceedings of the 2016 International Conference on Field-Programmable Technology*. FPT 2016. (2017)
123. Huang, Y. et al.: A high-efficiency FPGA-based accelerator for convolutional neural networks using Winograd algorithm. In: *Journal of Physics: Conference Series*, 6–8 March 2018 Location: Avid College, Maldives (2018)
124. Kim, J.H. et al.: FPGA-based CNN inference accelerator synthesised from multi-threaded C software. In: *International System on Chip Conference* (2017)
125. Li, Q. et al.: Implementing neural machine translation with bi-directional GRU and attention mechanism on FPGAs using HLS. In: *Proceedings of the Asia and South Pacific Design Automation Conference*, Tokyo, Japan, January 2019. ASP-DAC. (2019)
126. Ma, Y. et al.: An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks. In: *27th International Conference on Field Programmable Logic and Applications*. FPL 2017 (2017)
127. Xu, J. et al.: CaFPGA: an automatic generation model for CNN accelerator. *Microprocess. Microsyst.* 60, 196–206 (2018)

128. Zeng, S., et al.: An efficient reconfigurable framework for general purpose CNN-RNN models on FPGAs. In: International Conference on Digital Signal Processing, Shanghai, China, 19-21 November 2018. DSP. (2019)
129. Ma, Y. et al.: Scalable and modularised RTL compilation of convolutional neural networks onto FPGA. In: Fpl 2016 - 26th International Conference on field-programmable logic and applications, Lausanne, Switzerland, 29 August-2 September 2016 (2016)
130. Guan, Y., et al.: FP: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In: Proceedings - IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines, Napa, CA, 30 April-2 May 2017. FCCM 2017 (2017)
131. Alrawashdeh, K., Purdy, C.: Reducing calculation requirements in FPGA implementation of deep learning algorithms for online anomaly intrusion detection. In: Proceedings of the IEEE National Aerospace Electronics Conference, Dayton, OH, 27-30 June 2017. NAECON. (2018)

How to cite this article: Dhouibi M, Ben Salem AK, Saidi A, Ben Saoud S. Accelerating Deep Neural Networks implementation: A survey. *IET Comput. Digit. Tech.* 2021;15:79–96. <https://doi.org/10.1049/cdt2.12016>