FPGA-based Deep Learning Inference Accelerators: Where Are We Standing?

ANOUAR NECHI, LUKAS GROTH, and SALEH MULHEM, University of Lübeck, Germany
FARHAD MERCHANT, RWTH Aachen University, Germany and Newcastle University, United Kingdom
RAINER BUCHTY and MLADEN BEREKOVIC, University of Lübeck, Germany

Recently, artificial intelligence applications have become part of almost all emerging technologies around us. Neural networks, in particular, have shown significant advantages and have been widely adopted over other approaches in machine learning. In this context, high processing power is deemed a fundamental challenge and a persistent requirement. Recent solutions facing such a challenge deploy hardware platforms to provide high computing performance for neural networks and deep learning algorithms. This direction is also rapidly taking over the market. Here, FPGAs occupy the middle ground regarding flexibility, reconfigurability, and efficiency compared to general-purpose CPUs, GPUs, on one side, and manufactured ASICs on the other. FPGA-based accelerators exploit the features of FPGAs to increase the computing performance for specific algorithms and algorithm features. Filling a gap, we provide holistic benchmarking criteria and optimization techniques that work across several classes of deep learning implementations. This article summarizes the current state of deep learning hardware acceleration: More than 120 FPGA-based neural network accelerator designs are presented and evaluated based on a matrix of performance and acceleration criteria, and corresponding optimization techniques are presented and discussed. In addition, the evaluation criteria and optimization techniques are demonstrated by benchmarking ResNet-2 and LSTM-based accelerators.

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

Additional Key Words and Phrases: Deep learning, FPGAs, inference, accelerators

ACM Reference format:
Anouar Nechi, Lukas Groth, Saleh Mulhem, Farhad Merchant, Rainer Buchty, and Mladen Berekovic. 2023. FPGA-based Deep Learning Inference Accelerators: Where Are We Standing?. *ACM Trans. Reconfig. Technol. Syst.* 16, 4, Article 60 (October 2023), 32 pages.
https://doi.org/10.1145/3613963

This research was partially supported by the German Research Foundation (DFG) grant 403579441, project "Meteracom," and by the Federal Ministry of Education and Research (BMBF), project "NEUPA," grant 01IS19078.
Authors' addresses: A. Nechi, L. Groth, S. Mulhem, R. Buchty, and M. Berekovic, Institute of Computer Engineering, University of Lübeck, Ratzeburger Allee 160, Lübeck, 23562, Schleswig-Holsten, Germany; e-mails: {anouar.nechi, l.groth, saleh.mulhem, rainer.buchty, mladen.berekovic}@uni-luebeck.de; F. Merchant, RWTH Aachen University, Institute for Communication Technologies and Embedded Systems, Kopernikusstraße 16, Aachen, 52074, Nordrhein-Westfalen, Germany and Newcastle University, School of Engineering, Newcastle upon Tyne, NE1 7RU, Tyne and Wear, United Kingdom; e-mail: farhad.merchant@newcastle.ac.uk.

1936-7406/2023/10-ART60
https://doi.org/10.1145/3613963

1 INTRODUCTION

AlexNet [84], as a **Convolution Neural Network (CNN)**, has marked the milestone of the rapid development of **Deep Neural Networks (DNNs)**. Researchers have realized that DNN size is proportional to its accuracy, leading to the emergence of rapidly growing DNN architectures [148]. Consequently, DNN's outstanding results go hand-in-hand with their increase in computation and memory requirements. Although the DNN size is directly proportional to the accuracy for different subclasses of DNNs, this might not be the case for all DNN subclasses. For instance, a more efficient architecture such as Transformers might outperform CNN in accuracy at the same number of parameters as shown in Table 1. The increased parameter count of novel DNN architectures also leads to higher computational complexity, hindering them from being deployed efficiently on resource-constraint devices. A promising method to solve this problem is to employ neural architectures/operators that are more efficient on edge systems while not compromising accuracy. Models such as SqueezeNet [74], MobileNet [68, 69, 131], and ShuffleNet [103, 181] are lightweight models that rely on specialized operations to reduce the computational and memory requirements to accommodate resource-constraint devices.

1.1 Deep Learning Acceleration Hardware

Central Processing Units (CPUs) and **Graphical Processing Units (GPUs)** serve as general-purpose computing platforms for DNN inference, while **Field Programmable Gate Arrays (FPGAs)** and **Application Specific Integrated Circuits (ASICs)** can serve as dedicated hardware accelerators. The first generation of CPUs exhibit observable performance bottlenecks while running **Deep Learning (DL)** algorithms. Development of cutting-edge CPUs directly addressed the requirements of DL algorithms: In 2017, Intel released the Intel Xeon scalable processor to accelerate DNNs on lower-precision tasks [34, 130]. GPUs have been deployed first for enhancing computation of complex physical simulations and then for improving training and inference processes in DNNs [174]. It has been shown that when considering the tradeoff between accuracy and inference throughput, fixed-point representations, such as 8- and 16-bit, outperform floating-point representations commonly used during the training phase [76].

FPGAs are off-the-shelf programmable logic devices that provide a flexible platform for implementing custom hardware functionalities at lower development costs compared to ASIC design. FPGAs are deployed as computational accelerators in computing-intensive applications where high flexibility, fine-grained parallelism, and associative operations are needed [86]. DL accelerators leverage the flexibility of FPGAs for implementing architectures with a high degree of parallelism, resulting in higher performance. The adoption of software-level programming approaches in FPGAs tool flows, such as the OpenCL standard [111] and **High-level synthesis (HLS)** [32, 45], makes their use for DNN acceleration more attractive [115]. Further, ASICs are non-standard integrated circuits designed for a specific use or application. Since ASICs are custom circuits, they offer higher performance and energy efficiency than corresponding FPGA implementations. For ASIC-based DL accelerators, efforts have been made in both industry and academia, where representative ones include TPU [75, 79], ShiDianNao [38], and Eyeriss [26, 27]. These accelerators exploit various optimizations, such as quantization, systolic arrays, or dataflow architectures, which can be critical in power-sensitive applications and battery-operated devices [40, 106].

As such, FPGAs occupy the middle ground between CPU/GPU and ASIC implementations, resulting in higher reconfigurability and energy efficiency compared to CPU/GPU [108] and lower development costs than ASICs. Also, FPGAs operate at only a fraction of the clock speed achievable with CPUs, GPUs, and ASICs. The latter two, however, will typically expose similar levels of parallelism.

Table 1. State-of-the-art DNN Models

Convolution-based models			Transformer-based models			Lightweight models		
Model	**Top-1 (%)**	**Parameters (Mio.)**	**Model**	**Top-1 (%)**	**Parameters (Mio.)**	**Model**	**Top-1 (%)**	**Parameters (Mio.)**
DenseNet-121 [70]	75	8.1	MobileViT-S [105]	78.4	5.6	SqueezeNet [74]	60.4	1.24
Xception [28]	79.0	22.9	TinyViT [167]	86.5	21	MobileNet-V2 [131]	72	3.4
EfficientNet-B5 [147]	83.6	30.6	BoTNet-T3 [141]	81.7	33.5	ShuffleNet [181]	71.5	3.4
NasNetLarge [183]	82.5	88.9	BoTNet-T7 [141]	84.7	75.1	GhostNet [59]	73.9	5.2
ConvNextLarge [101]	86.3	197	Florence CoSwin-H [176]	90.5	893	EfficientNet-B0 [147]	77.1	5.3

Table 2. Qualitative Comparison among State-of-the-art Surveys

Survey	**Year**	**Acceleration metrics**		**Hardware Platforms**				**Investigated models**
		Throughput	**Energy efficiency**	**CPU**	**GPU**	**FPGAs**	**ASIC**	
Shawahna et al. [135]	2018	✓	✓	✗	✗	✓	✗	CNN
Reuther et al. [129]	2019	✓	✓	✓	✓	✓	✓	CNN
Feng et al. [42]	2019	✓	✓	✗	✓	✓	✗	CNN
Wang et al. [162]	2019	✓	✓	✗	✗	✓	✗	CNN, RNN
Guo et al. [57]	2019	✓	✓	✗	✓	✓	✗	CNN, RNN
Li et al. [97]	2020	✗	✗	✗	✗	✓	✗	CNN
Capra et al. [20]	2020	✓	✓	✗	✗	✗	✓	CNN
Chen et al. [24]	2020	✓	✓	✓	✗	✗	✓	CNN, RNN
Wu et al. [168]	2021	✓	✓	✗	✗	✓	✓	CNN, RNN
Ours	2023	✓	✓	✗	✗	✓	✗	CNN, RNN Autoencoders, Transformers

As presented in Table 2, several works were published exploring the state of hardware acceleration platforms. Some works aim at covering various aspects of the acceleration platforms and compare them based on many criteria. For instance, the hardware architectures of **Tensor Processing Units (TPUs)**, CPUs, GPUs, and FPGAs were only compared in Reference [132]. Performance and power consumption numbers of machine learning accelerators were presented and analyzed in Reference [129] based on CPUs and GPUs. In Reference [116], the performance and energy consumption of state-of-the-art GPUs were evaluated. Other works concentrate mainly on accelerators for **Artificial Intelligence (AI)** applications; for instance, AI accelerators for edge environments were systematically investigated in Reference [94]. However, the required hardware resources for such accelerators were not considered. In Reference [63], the effect of AI development on server design was analyzed.

Furthermore, multiple recent surveys focus on the hardware architecture of ASIC-based accelerators more than FPGA-based accelerators [20]. Other surveys provide few details about DNN acceleration techniques [63, 129]. Recently, there has been special emphasis on FPGA-based accelerators. For instance, a wide range of FPGA designs for AI was reviewed and reported on in Reference [97]. **Digital Signal Processor (DSP)** modules, adaptive logic modules, configurable logic blocks, and memory modules were discussed without details on DNN acceleration techniques. In Reference [135], FPGA-based accelerators with special emphasis on CNN were presented. The work shows how FPGAs efficiently accelerate CNN by maximizing the parallelism of operations. The hardware resource utilization of several FPGA-based accelerators was reported in more detail. In Reference [42], computer vision algorithms were investigated. A summary of different CNN models was presented, including AlexNet [84], VGGNet [138], GoogleNet [145], DenseNet [70], ShiftNet-A [165], and FE-Net [22]. However, the performance comparison of FPGA-based CNN accelerators was introduced without corresponding hardware resource utilization.

Prior works have covered CNN inference accelerators while ignoring DNN inference accelerators [20, 42, 97, 129, 135]. Other surveys have been published targeting either different technologies such as **Resistive Random Access Memory (RRAM)**-based accelerators [24] or focusing on various issues of FPGA-based accelerators such as reliability [107]. Such approaches are out of the scope of this work and will, therefore, not be evaluated.

1.2 Motivation, Contribution, and Article Organization

The motivation of this article is not just to provide a comprehensive overview of the state-of-the-art in the domain of FPGA-based DL inference accelerators but also sufficient metrics and techniques for evaluation and optimization. To the best of our knowledge, this work is the first study that covers DNNs in general, not only a subclass of DNNs such as CNN. The individual contributions of this article are as follows:

— We present an overview of the most relevant DNN architectures that have been the focus of research for hardware acceleration during the last years (Section 2). Correspondingly, we highlight key FPGA features and explain their specific suitability for DNN acceleration (Section 3).
— We detail recent DNN acceleration strategies and highlight their influence on an FPGA-based DNN inference accelerator regarding throughput, resource usage, and energy efficiency (Section 4).
— We introduce and discuss metrics and acceleration criteria that can be uniformly applied to evaluating FPGA-based DNN accelerators. To emphasize the validity of the proposed acceleration criteria, we implement a ResNet-2 and an LSTM model on the same FPGA with various pruning, quantization, and parallelization configurations (Section 5).
— To the best of our knowledge, we present the largest collection of performance results from FPGA-based DNN inference accelerators recently published in academia.

2 DEEP LEARNING PRELIMINARIES

For decades, classical **Machine Learning (ML)** has been limited in its ability to process natural data. ML requires considerable domain expertise to design a feature extractor that transforms raw data into a suitable internal representation for detecting and classifying patterns in the input data. In response to this challenge, representation learning was proposed. Several methods feed the machine with raw data and extract the features for dynamic and automatic detection, or classification [12].

The following focuses specifically on accelerating DNNs. As proposed by Reference [24], DNNs shall refer to NNs composed of multiple layers, where each layer applies a transformation to the input- or intermediate data, respectively. Here, the operation $f_i(x_{i-1})$ applied by layer i of the DNN to the result of the previous layer x_{i-1} can be any arbitrary operation used for building NNs, such as convolution, fully connected, pooling, and so on, or a stack of operations such as a residual- or dense block. The inference result obtained with a l-layer DNN can then be defined as applying the composition operation ($\circ$) of the individual layer-functions to the input data x_{in} [24]:

$$f(x_{\text{in}}) = f_l \circ f_{l-1} \circ f_{l-2} \circ \cdots \circ f_2 \circ f_1(x_{\text{in}}).$$

In this section, the structural elements of DNNs as well as their low bit-width adaptations uniquely suited for hardware acceleration are detailed and explained. Furthermore, the evolution of DNNs is presented to point to the increasing challenges faced by deep learning and, subsequently, deep learning accelerators over the years.

2.1 Convolutional Neural Networks

In DL, CNNs are the most famous and commonly employed algorithm [3, 91, 161]. Compared to its predecessors, the main benefit of CNN is that it learns suitable filter kernels for feature extraction by providing examples from the target domain [54]. Therefore, CNNs have been widely used in a variety of applications, including computer vision [41], speech processing [118], facial recognition [93], and so on. Similar to a conventional neural network, the structure of a CNN was

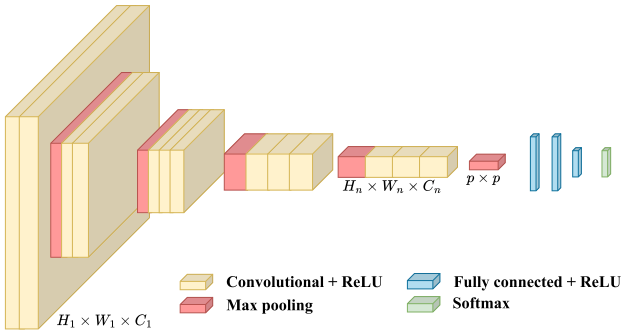

Fig. 1. VGG-like CNN architecture for image classification.

inspired by the visual cortex, which consists of a complex sequence of brain neurons interacting with the optical nerve. Furthermore, three key benefits of CNN were identified: equivalent representations, sparse interactions, and parameter sharing [88]. In contrast to conventional **fully connected (FC)** networks, shared weights and local connections in the CNN are employed to fully use 2D input-data structures such as image signals. In addition, this operation utilizes fewer parameters compared to FC layers, which both simplifies the training process and speeds up the inference by requiring fewer operations. A common type of CNN includes several convolution layers with nonlinearities in between, preceded by intermittent pooling (sub-sampling) layers and FC layers at the end. Figure 1 illustrates a VGG-like CNN architecture for image classification. It consists of five convolutional blocks together with three FC layers and Softmax as an activation function for the last layer.

The input X of each convolutional layer i is organized in three dimensions: height, width, and depth, or the number of channels denoted as H_i, W_i, and C_i, respectively. Each convolutional layer contains c_k filter kernels K, which have three dimensions $(h_k \times w_k \times C_i)$. Here, (h_k, w_k) are generally much smaller than (H_i, W_i). In addition, the kernels K are the basis of the local connections with parameters for biases b^k and weights W^k. For generating c_k output feature maps h^k, each kernel is convolved with the input. The convolutional layer performs a dot product of its input and the weights, where the inputs are sections of size $(h_k \times w_k \times C_i)$ of the input data X. Next, a nonlinear activation function $\sigma(.)$ is applied to the convolution-layers output to obtain the feature maps $h^k = \sigma(W^k * X + b^k)$.

Often, convolutional layers or blocks are followed by down-sampling layers called pooling layers. These layers reduce the network parameters, which accelerates the training process and reduces the spatial dimension. This allows subsequent convolution layers to correlate features that were initially far apart in the input data using smaller filter kernels. For all feature maps, the pooling function, such as maximum or average pooling, is applied to an adjacent area of size p × p, where p is the pooling size [5]. Finally, the FC layers receive the mid- and low-level features to create the high-level abstraction, representing the last-stage layers in a typical CNN for image classification.

2.1.1 Residual Networks. **Residual Networks (ResNets)** came as a response to the vanishing or exploding gradient problem. This problem usually appears when the number of layers increases, leading to the error rate growing rapidly. Such a problem causes the gradient to become small or too large, resulting in too-aggressive parameter updates during backpropagation, prohibiting effective learning. ResNet was first proposed in Reference [64] to design ultra-deep networks that avoid the vanishing or exploding gradient problems. This model introduces shortcuts inside the

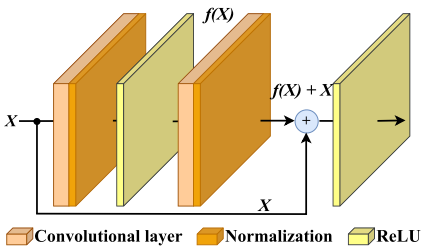

Fig. 2. The residual block consists of a stack of layers representing the original mapping $f(X)$ and a skip connection X. The desired residual mapping $h(X)$ is obtained by adding them together [64].

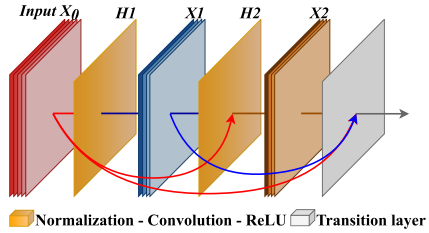

Fig. 3. In a Dense Block, each layer gets input from previous ones, combines it with its own, and sends it to subsequent layers. The transition layer changes feature-map sizes through convolution and pooling for the next block [70].

network layers to enable cross-layer connectivity, thus allowing the gradient to flow through these connections uninterrupted during training. These shortcuts are parameter-free, data-independent, and avoid the problem of gradient diminishing by deconstructing the deeper network into multiple shallower networks. The core idea of ResNets is to define the desired underlying mapping $h(X)$ (Residual mapping) as $h(X) = f(X) + X$, where $f(X)$ is the original mapping and X is a "skip-connection," as shown in Figure 2.

2.1.2 DenseNets. **Densely connected convolutional networks (DenseNets)** was proposed right after ResNet to solve the vanishing gradient problem [70]. DenseNet employs cross-layer connectivity in an improved approach to address the issue of weights number in ResNet [85]. Each layer is connected to all layers in the network using a feed-forward approach, as illustrated in Figure 3. In CNNs, there are l connections between the previous layer and the current layer, while there are $l(l+1)/2$ direct connections in DenseNet. As a result, DenseNet exhibits the influence of cross-layer depthwise-convolutions. Thus, the network can distinguish between the added and the preserved information, since DenseNet concatenates the preceding layers' features instead of adding them. In addition to the increased number of feature maps, DenseNet is parametrically expensive due to its narrow layer structure. The direct admission of all layers to the gradients via the loss function enhances the information flow across the network. Moreover, this includes a regularizing impact, which minimizes overfitting on tasks with small training sets.

2.2 Recurrent Neural Networks

RNNs are a commonly employed algorithm in DL [66, 78]. An RNN is a neural network that simulates a discrete-time dynamical system with an input x_t, an output y_t, and a hidden state h_t [121]. The system is defined by:

$$h_t = \phi_h(W^T h_t + U^T x_t), \tag{1}$$

$$y_t = \phi_o(V^T h_t), \tag{2}$$

where W, U, and V are the transition, input, and output matrices, respectively, and ϕ_h and ϕ_o are element-wise nonlinear functions. RNNs target sequential inputs such as time series x_t and take advantage of such sequential data structures to achieve high accuracy. This feature is fundamental for various applications, mainly speech processing [9] and **Natural Language Processing (NLP)** [77]. For instance, it is essential to understand the context of a sentence to determine the meaning of a specific word. A key challenge with RNNs is their high computation and latency overhead. RNNs work on variable-length input/output sequences, and their inference latency depends on the length of the inputs, as every new result depends on all the previous ones

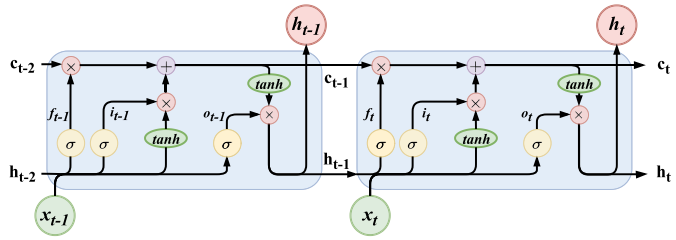

Fig. 4. An LSTM module. The parameters between the neurons are omitted for better readability.

(Equations (1) and (2)). This leads to data dependencies across timesteps. Thus, the sequential nature of RNN makes the parallelization of the computation across the entire series a very challenging task. Other challenges of this approach are RNN's susceptibility to exploding and vanishing gradient problems [51]. Specifically, reduplications of several large or small derivatives throughout the training phase may cause the gradients to explode or decay exponentially.

Long Short-Term Memory (LSTM), a subset of RNNs, was introduced to handle the vanishing gradient problem. Unlike conventional RNNs, the hidden layers in LSTMs are complex structures interacting in a particular way, allowing them to handle long-term dependencies, as shown in Figure 4. The core idea behind the LSTM concerns the cell state c_t, which serves as a memory and information carrier. Moreover, the information to add or remove from the cell state is regulated by structures called gates, which are usually composed of a *Sigmoid* layer and a pointwise multiplication operation. Supposing that the activation functions in the nonlinear layers are all *tanh* functions, the forget gate f_t, input gate x_t, cell state c_t, output gate y_t, and hidden state h_t, at time t are calculated as follows:

$$f_t = \sigma(W^f[h_{t-1}, x_t] + b^f), \tag{3}$$
$$i_t = \sigma(W^x[h_{t-1}, x_t] + b^x), \tag{4}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W^c[h_{t-1}, x_t] + b^c), \tag{5}$$
$$o_t = \sigma(W^o[h_{t-1}, x_t] + b^o), \tag{6}$$
$$h_t = o_t \odot \tanh(c_t), \tag{7}$$

where W^* and b^* are learnable parameters and $*$ denotes the superscripts including f, i, and o. The cell states need to be updated together with the hidden states each time. The output of the LSTM module at time t is decoded from the hidden states as in the classic RNN. Indeed, applying the forget gate is the key to learning more effectively the sequential information context [47].

2.3 Autoencoders

Autoencoders comprise an encoder-decoder architecture where the encoder compresses a fixed-size input x_{in} into a fixed-size latent representation z of lower dimension. The main idea of an *Autoencoders* is that z should retain all information present in x_{in}. This would allow the decoder to reconstruct x_{in} from z. The encoder and decoder may include any layers, such as fully connected, convolution, or LSTM. *Autoencoders* usually employ convolution layers in the encoder and decoder stacks in image processing. Convolution operations use a sliding window with fixed kernel size $(h_k \times w_k \times C_i)$. Keeping h_k and w_k small makes it impossible to correlate features in the input data that are spatially far apart. Here, as previously mentioned in Section 2.1, an encoder can be used to compress a fixed-sized input x_{in} into a fixed-size intermediate (or latent) representation z of the data. The encoder is built from convolution layers followed by pooling layers that are used

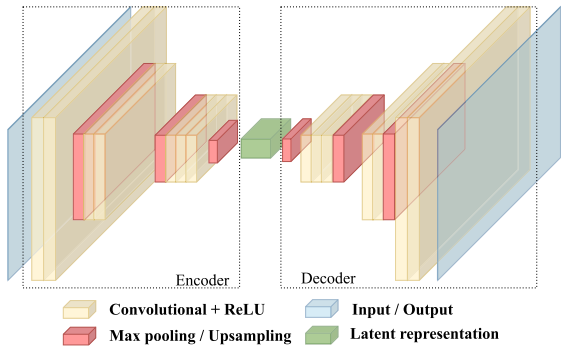

Fig. 5. Autoencoder architecture that reconstructs an input image based on the latent representation z.

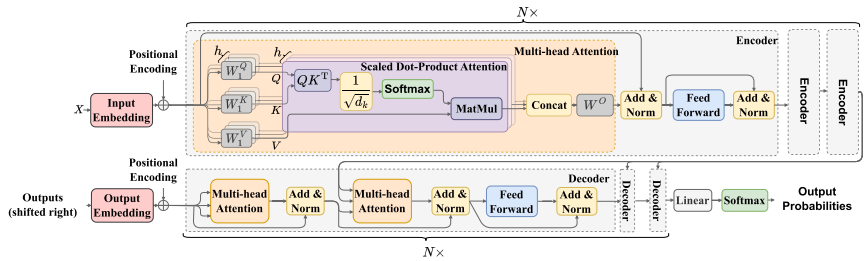

Fig. 6. The Transformer DNN architecture consisting of N encoders and decoders. Both contain the *multi-head attention mechanism* with h-heads.

to iteratively reduce the spatial dimension while increasing the number of filter channels, thus increasing the feature dimension.

In contrast to Section 2.1, *Autoencoders* do not use the latent representation z for classification but, instead, apply a decoder to reconstruct x_{in} or a different representation of it. The decoder applies de-convolution using transpose convolution layers to restore the spatial dimension according to the target application by iteratively decreasing the feature dimension and increasing the spatial dimension. For instance, an *Autoencoder* might be trained to compress an input image x_{in} using the encoder and then restore the same image using the decoder based on information encoded in z.

Figure 5 shows such an *Autoencoder* for image reconstruction. This concept can then be extended to various applications, including image-denoising [53], segmentation [112], or up-scaling [177].

2.4 Transformers

Recently, Transformer architectures have partially replaced conventional RNN and LSTM models, especially for NLP applications. Transformers enable parallel processing of an input X, allowing for faster training and the ability to retain context over longer sequences compared to LSTM-based models [151]. As illustrated in Figure 6, Transformers are based on an encoder-decoder architecture that employs the so-called *Attention mechanism* instead of convolutions or recurrences. *Attention* is inspired by the visual attention system of human perception, meaning the entire visual field is not processed all at once. Instead, *Attention* is restricted to selective regions of interest, while regions that do not contribute to the scene's context are disregarded. Thus, in the context of DNNs, Transformers do not need to compress X into a latent representation, since they can selectively attend to different parts of a sequence. The *Scaled Dot-product Attention* mechanism

Table 3. FPGA-based Accelerator Results for BERT and RoBERTa

Accelerator	Model	Parameters (Mio.)	Platform	Throughput (FPS)	Power (W)	Density (FPS/DSP)
FTRANS [89]	$\text{RoBERTa}_{\text{BASE}}$ [99]	125	Virtex VCU118	101.79	25.13	0.0156
NPE 8-bit [81]	$\text{BERT}_{\text{BASE}}$ [37]	110	Zynq Z-7100	135.14	20.00	0.0669
NPE 16-bit [81]	$\text{BERT}_{\text{BASE}}$ [37]	110	Zynq Z-7100	73.69	20.00	0.0365

Table 4. XNOR Logical Operation on the Binary Encodings

Encodings (values)		XNOR
0 (−1)	0 (−1)	1 (+1)
0 (−1)	1 (+1)	0 (−1)
1 (+1)	0 (−1)	0 (−1)
1 (+1)	1 (+1)	1 (+1)

used in Transformer neural networks is composed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\text{T}}}{\sqrt{d_k}}\right)V,$$

with $Q = XW^Q$ the *query*, $K = XW^K$ the *key*, $V = XW^V$ the *value*, which are obtained by multiplying each element x_i of the input X with the fixed weights matrices obtained during training W^Q, W^K, W^V and stacking them up to form the respective matrix. To stabilize the gradients, the result inside the softmax function is scaled with the dimension of the key vectors $\sqrt{d_k}$. This results in a score expressing the correlation of each word regarding all other tokens in X. In the Transformer architecture, each (W_i^Q, W_i^K, W_i^V) pair forms the basis for a single so-called *attention head*; h heads are then concatenated to form the *multi-head attention* mechanism as per:

$$\text{head}_i = \text{Attention}\left(XW_i^Q, XW_i^K, XW_i^V\right)$$

$$\text{MultiHead}\,(Q, K, V) = \text{concat}\,(\text{head}_1, \ldots, \text{head}_h)\,W^O.$$

This allows Transformers to attend to information from different representation subspaces at different positions [139, 151]. The benefits described regarding parallelization over LSTM models can unlock high potential for the design of inference accelerators. As illustrated in Table 3, recent Transformer-based NLP models have millions of parameters, making it infeasible to store them in the on-chip BRAM of an FPGA [123]. Consequently, FPGA-based Transformer accelerators require efficient memory accesses during inference and efficient processing systems for the large matrix multiplications and non-linearities, such as the softmax, applied over high-dimensional data.

2.5 Binarized Neural Networks (BNNs)

Courbariaux et al. [30, 71] developed the BNN methodology used by most network binarization techniques. The idea of BNN is to constrain both weights and activations to (+1) and (−1). The binarization method can be done stochastically or deterministically using the *sgn()* function.

$$W_{\text{Binary}} = \text{sgn}(W_{\text{Real}}) \tag{8}$$

Binarization of both weights and activations in BNNs reduces memory requirements and computational complexity by allowing the use of bitwise operations. The dot product of weights and activations can then be reduced to binary bitwise operations. The signed binary numbers are represented as 0 for −1 and 1 for +1. As shown in Table 4, applying an XNOR logical operation on the binary encodings is equivalent to performing multiplication on the binary values.

Moreover, a dot product requires accumulating all the products between values. Although XNOR may perform bitwise multiplication, accumulating requires a summation of the XNOR results. The accumulation can be accomplished by counting the number of 1's (*popcount*) in a group of XNOR products, multiplying by two (*left-shift*), and subtracting the total number of bits resulting in an integer value. The XNOR, popcount, and shift operations are much simpler to compute than floating-point or fixed-point multiplication and accumulation, leading to faster execution and/or less hardware resource utilization [137].

2.6 Ternary Neural Networks (TNNs)

Ternary Neural Networks (TNNs) constrain the full precision values to the ternary space {−1, 0, 1}, resulting in an extra zero state compared with BNNs. Nevertheless, TNNs can also be implemented as efficiently as BNNs. As demonstrated in GXNOR-Nets, for a multiplication operation in TNNs, when one of the weights and activation is zero or both are zero, the corresponding computation unit rests until both are enabled, and the XNOR compute unit is triggered [36]. A TNN represents the rational weights and/or activations with ternary values [4]. Formally, the quantization can be expressed as:

$$T_x = \begin{cases} +1 & \text{if} \quad x > \Delta \\ 0 & \text{if} \quad |x| \leqslant \Delta \\ -1 & \text{if} \quad x < -\Delta \end{cases} \qquad \text{where} \quad \Delta = \frac{0.7}{n} \sum_{i=1}^{n} |x_i|, \tag{9}$$

where x indicates rational parameters including weights and activations, T_x denotes ternary values after the quantization on x. T_x is usually obtained by applying a thresholding function Δ, which denotes a fixed threshold used for quantization obtained based on the rational parameters x, with n being the number of parameters in x.

$$z = T_W \odot T_A \tag{10}$$

Subsequently, with the ternary weights W and activations A, the multiplications in the forward propagation can be reformulated as shown in Equation (10).

3 DNN ARCHITECTURE CHALLENGES FOR FPGA ACCELERATORS

DL algorithms often benefit from many-core hardware and high bandwidth memory. However, DNN hardware acceleration still faces two main challenges [173]: (1) increasing the speed of data transfer and process and (2) enhancing the DNN performance. This section details the FPGA architecture and its main building blocks together with the advantages for DNN acceleration followed by an analysis and discussion about the current challenges of FPGA-based DNN accelerators.

3.1 DNN Architecture Evolution

DNN models have grown larger and deeper to achieve higher accuracy. Figure 7 illustrates the direct proportionality between DNN model accuracy and the number of trainable parameters. This increase in DNN model size led to a remarkable development of DNN architectures [144] and increased accuracy, albeit requiring more computing and memory resources. Additionally, the large volume of data and operations poses a major challenge for computing platforms, mainly for edge devices with limited computing resources. For instance, an accuracy-oriented model such as ResNet-50 has 25 million parameters and 5.71 billion operations per inference. This limits the feasibility of deploying the model on resource-constrained devices. These hardware resource limitations can be alleviated by using the following techniques: (1) compression techniques that reduce the memory footprint at the cost of accuracy, such as pruning or quantization (see Section 4.1), and

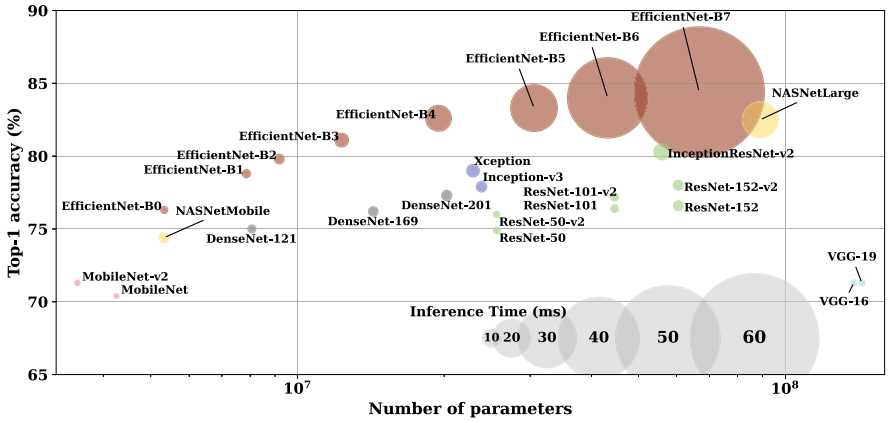

Fig. 7. Top-1 accuracy vs. the number of parameters and inference time on Nvidia A100 GPU.

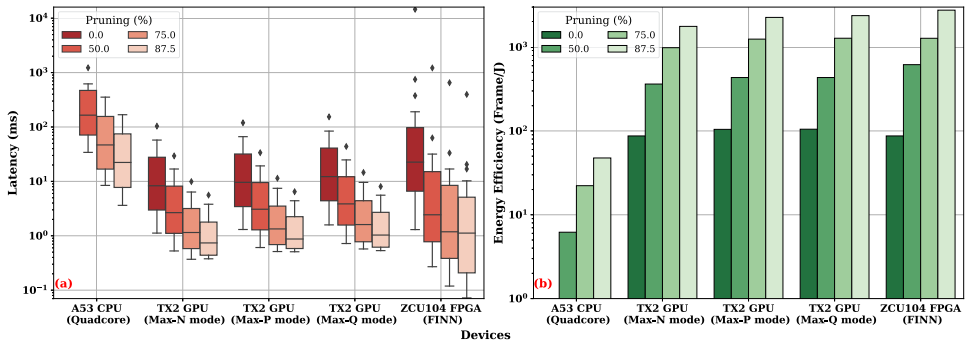

Fig. 8. CPU, GPU, and FPGA (a) latency and (b) energy efficiency comparison (based on QuTiBench [15]).

(2) optimized hardware designs that reduce the computational complexity to increase throughput (see Section 4.2). Consequently, achieving a balance between an NN's model size and accuracy is still challenging.

3.2 Why are FPGAs well-suited for DNN acceleration?

FPGAs consist of **configurable logic blocks (CLBs)** with programmable interconnects to customize operations for a specific application. This flexible configurability allows an FPGA to accommodate and ensure design changes or even new applications. The FPGA architecture includes **Input/Output (I/O)** blocks and **Digital Signal Processing (DSP)** blocks. Furthermore, the input and output paths contain edge-triggered **Flip-Flops (FFs)**, providing the user an interface from the outside world to the FPGAs internal architecture. DSP cells, as one of the main building blocks, carry out **Multiply-Accumulate (MAC)** operations much faster, such as the case of DSP48E2 slices used by Xilinx [170]. The DSP cells and flexible programmability provide advantages of FPGAs over CPUs and GPUs for DNN acceleration.

Figure 8 shows several computing platforms used to execute and accelerate CNN inference trained on the CIFAR-10 dataset [15]. The results indicate low latency and high energy efficiency of FPGAs in accelerating CNN models compared to the A53 CPU and the TX2 GPU. This emphasizes the applicability of FPGAs for DNN acceleration.

3.3 Challenges & Solutions of FPGA-based DL Accelerators

In the following, several challenges of FPGA-based DNN accelerator designs are addressed:

3.3.1 HDL vs. HLS vs. DSL. **Hardware Description Language (HDL)**-based design delivers high efficiency and throughput but requires significant knowledge of the AI model and FPGA architecture [17, 107]. Nowadays, HDL design is experiencing a shift to higher abstraction levels by embedding the low-level design features of native HDLs into functional programming languages such as Haskell and Scala [6, 7, 95] or by relying on imperative paradigms, such as those offered by Java and Python [10, 104, 146]. This aims to make the FPGA ecosystem more accessible for ML engineers. Furthermore, the reusability and validation of components across hierarchical and decoupled designs are benefits of these new paradigms. HLS-based design allows for even faster development through high-level abstraction. HLS may also provide features such as automated partitioning and pipelining of DNN models for multiple FPGAs [31]. For instance, hls4ml [39] and Finn [16, 150] are two HLS-based packages for creating FPGA-based AI inference accelerators. Nevertheless, the HLS-generated designs may not be as well optimized as HDL-based designs for hardware efficiency. **Domain-specific languages (DSLs)** [43] provide a higher abstraction level than Verilog and VHDL, which reduces the required hardware expertise and learning curve for a target domain. Additionally, DSLs enable quick and easy development of portable code for multiple architectures and highly optimized implementations. Some DSLs concentrate on a given application class by implementing specific constructs and abstractions that ease the development of efficient code for a given domain. Others shift the focus from the application level to the architectural one. They implement a particular architectural template and offer features and constructs to support classes of algorithms.

A qualitative evaluation of HDLs, HLS, and DSLs tools shows that HDLs offer the best flexibility, performance, and resource efficiency, but at the cost of design, verification time, and conciseness. DSLs boost productivity and conciseness and deliver good design quality. HLS represents a good tradeoff between HDLs and DSLs. [140]

3.3.2 Hardware Resource Limitations. Internal FPGA memory resources are limited and may be insufficient for large DNN models in real-life classification or recognition tasks. For instance, AlexNet has around 60 million 32-bit floating-point parameters, yielding 240 MB for storing the weights and requires approximately 1.5 GOPs for processing each input image [143]. Such required resources cannot be provided by most modern FPGAs. Therefore, the model weights should be stored in external memory and transferred to the FPGA during inference.

3.3.3 Careful Design and Optimization. The different layers of a DNN model have various characteristics, resulting in additional parallelism and memory access requirements. For instance, each convolutional layer in a VGG-16 requires 2 to 9 GOPs and has 0.04 to 7.08 million weight parameters. However, FCLs have an order of 0.01 to 0.21 GOPs, while the number of weights is in the order of 4.10 to 102.76 million. Therefore, the design of a DNN accelerator should be optimized, meet different layers' requirements, and be flexible to maximize each NN layer's performance.

To address these requirements and allow for efficient DNN implementations, analysis of FPGA resources is needed, together with reusing computational resources and storing partial results in internal memories. The order of operations and the desired parallelism significantly impact data transfer and computational resource usage when implementing DNN models on FPGAs. One solution is to schedule operations. This results in a significant reduction in external memory access and internal buffer sizes. Another solution is to reduce the precision for representing the weights. This can decrease external memory bandwidth requirements with minimal impact on quality while simultaneously resulting in better energy efficiency.

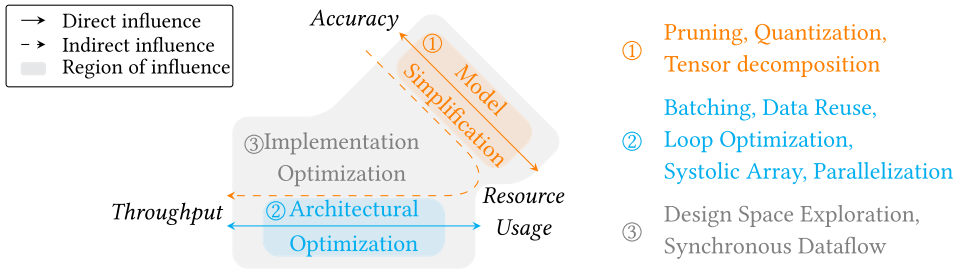

Fig. 9. Optimization and simplification techniques used in the hardware mapping process of AI models and their influence. Solid arrows indicate direct influence, dashed arrow shows indirect influence.

4 FPGA-BASED DNN ACCELERATION STRATEGIES

The primary properties of a DNN hardware accelerator are *Accuracy*, *Resource Usage,* and *Throughput,* as illustrated in Figure 9. The acceleration strategies include model simplification, architectural, and implementation optimization. When mapping an DNN model to the FPGA, model simplification directly affects accuracy and resource usage, while it indirectly influences throughput through its effects on resource usage. Architectural Optimization directly impacts resource usage and throughput without affecting accuracy.

This section explores FPGA-based DNN acceleration strategies by following the hardware mapping flow. The DNN model can first be simplified, gaining performance potential at the expense of accuracy through *Model Simplification.* Next, *Architectural Optimizations* can be applied to maximize density. Then, *Implementation Optimization* is employed to find suitable hyper-parameters used in Model Simplification and Architectural Optimization.

4.1 DNN Model Simplification

DNN Model Simplification aims to increase the DNN model's sparsity while maintaining accuracy. Increasing the sparsity leads to reducing the storage and computational requirements, thus achieving higher performance potential. This is achieved by quantization, pruning, or tensor decomposition methods.

4.1.1 Pruning. Pruning refers to the process of compressing a DNN model by removing unnecessary parameters [14]. The benefit of pruning is that it can be applied after the model has been trained at the cost of accuracy. To retain the accuracy, an additional sparsity promoting loss can be integrated into the training process, allowing for, in some cases, more than 70% of parameters to be pruned without degrading the accuracy compared to the accuracy of the baseline network [100]. There are different methods for pruning NNs: (1) *weight pruning* sets individual parameters to zero, thus sparsifying the NN while conserving the NN architecture, (2) *neuron pruning* reduces the size of the network architecture by removing entire neurons together with all incoming and outgoing connections, (3) *channel pruning* removes entire feature map channels from convolutional layers [65, 110]. All the methods mentioned above require training the full dense model first and then adding additional rounds for fine-tuning until the desired sparsity is achieved.

In previous works, Page et al. [117] presented SPARCNet, a hardware accelerator for sparse CNNs implemented on an Arty7 FPGA board. The VGG-D DNN model achieved almost 94% accuracy on the CIFAR-10 dataset [83] with 16-bit fixed-point weights, resulting in a design achieving a throughput of 31.79 GOP/s at 100 MHz with a dynamic power of 1.82 W and energy efficiency of 17.46 GOP/j. The pruning techniques implemented within the accelerator reduced the computation, memory access, and data transfer by skipping zero-valued parameters, allowing the design to

achieve high performance and energy efficiency with negligible accuracy loss. In Reference [50], row-balanced pruning is applied to an LSTM model for speech recognition executed on a single processing unit. Compared to the state-of-the-art work, the proposed architecture and pruning algorithm provided, on average, 128% improvements in energy efficiency. In Reference [60], a similar pruning method is applied to an LSTM model and implemented on Xilinx FPGA XCKU060 executed on multiple processing units. Weight sparsity is exploited by allowing faster **Processing Units (PUs)** to obtain data from the activation queue and continue working without waiting for the others. The results show that the proposed design achieves 43× and 3× higher throughput and 40× and 11.5× higher energy efficiency than CPU and GPU platforms, respectively.

4.1.2 Quantization. Quantization is a method for reducing both the memory footprint as well as the computational complexity of the model by reducing the bit width of the parameters [18, 182] at the cost of accuracy. The floating-point representations of the initial model's parameters are approximated with lower bit-width fixed-point representations, possibly down to 1- or 2-bit as in BNN and TNN, respectively (see Sections 2.5 and 2.6). There are two approaches to quantization: **Post-Training Quantization (PTQ)** [8, 166, 172] and **Quantization-Aware Training (QAT)** [29, 72, 76, 120]. As the name suggests, PTQ uses floating-point parameters and activations during training and then applies quantization after training. In QAT, the quantization error is considered part of the training loss during training. Although PTQ is easier to implement, QAT improves the quantized model's accuracy.

Reducing the number of bits for DNN parameters lowers the computation requirements and increases the model's performance and scalability. Most state-of-the-art FPGA designs rely on quantization to replace the 32-bit floating-point arithmetics with fixed-point arithmetics. The community has widely employed 16-bit fixed-point arithmetics. For instance, designs in References [55, 92, 125, 169, 179] have employed the 16-bit fixed-point precision to accelerate CNNs on an FPGA. Others in References [19, 46, 160] leveraged the same precision to accommodate RNNs on FPGA devices. Some designs adopted even lower precision, such as 12-and 8-bit precision [56, 98, 126, 171]. In Reference [125], Qiu et al. adopt a dynamic quantization scheme for CNNs using 8 and 4 bits for convolutional and dense layers, reducing storage and saving bandwidth compared to 8 bits for both layers. Additionally, many researchers have investigated DNN parameter binarization to reduce the model storage space and increase scalability. A 2-Bit VGG16 accelerator proposed by Yonekawa et al. [175] achieved a throughput of 460.8 GOP/s at 22 W, yielding an energy efficiency of 20.94 GOP/s/W. Although BNNs and TNNs suffer from accuracy loss, many designs explore the benefit of using 1- or 2-bit precision for computation to achieve real-time inference capabilities [36, 44, 48, 49, 182].

4.1.3 Tensor Decomposition. Tensor decompositions are techniques that factorize multi-dimensional tensors into smaller matrices. Various matrix factorization techniques have been proposed for accelerating DNNs, including **Canonical Polyadic Decomposition (CPD)** [67], **Batch Normalization Decomposition (BMD)** [73], Tucker-2 Decomposition [61], and **Singular Value Decomposition (SVD)** [142]. According to Kolda et al. [82], most factorization techniques can be applied to DNNs. However, some may not offer the best tradeoff between accuracy and computation complexity.

Among the existing techniques, CPD and BMD have demonstrated the best performance in terms of accuracy, with CPD compressing the DNN more than BMD, thus resulting in greater acceleration. However, CPD may lead to unsolvable optimization problems, rendering factorization impossible [45], whereas BMD is known to be a stable factorization technique. Recently, authors in Reference [90] proposed a tensor decomposition-based framework for accelerating DNNs, which yielded significant improvements in computation efficiency by 1.8× and 4.7× on FPGA compared

to CPU and GPU implementations. Nevertheless, the framework involves reiterations of decomposing and fine-tuning to learn an accurate network structure. In Reference [128], the authors investigated the impact of various compression techniques across different acceleration platforms. They applied Tucker decomposition and 8-bit quantization to a ResNet-50 model. The decomposition process led to the formation of reduced-size matrices, enabling FPGA to achieve higher computation parallelism. Consequently, results indicate a notable reduction in runtime of 90% on the Xilinx Virtex-7 FPGA compared to the baseline model. Additionally, the proposed decomposition techniques achieved high compression levels of 96.5%, 96.7%, and 93% on ResNet-18, VGG-11, and ResNet-50, respectively, while maintaining acceptable accuracy levels.

4.2 Architectural Optimizations

After the model has been simplified, Architectural Optimizations target the hardware design to achieve higher performance.

4.2.1 Batching. Batching is the procedure of simultaneously processing multiple input data (e.g., images, frames) to improve system throughput. This approach transforms a Matrix-Vector multiplication into a Matrix-Matrix multiplication. Due to the high number of parameters in modern NNs (see Section 3.1), it is impossible to permanently store them in the FPGA, necessitating off-chip memory accesses. Furthermore, loading all weights whenever a new input sample needs to be processed is inefficient.

Batching aims to maximize throughput by buffering and reusing a subset of weights and applying them to all samples in a batch. After a set of weights has been used in a batch, the next set is loaded. As shown by Shen et al. [136], batching effectively reduces the memory bandwidth requirements. In their evaluation of AlexNet, VGGNet-E, and GoogleNet, they report an average reduction in peak bandwidth by a factor of 2.26× and 2.5× at 32- and 16-bit, respectively, compared to previous works while maintaining the same throughput.

4.2.2 Parallelization. Three NN parallelization strategies exist: data parallelism, model parallelism, and layer pipelining. Data parallelism is a strategy that allows working on different sections or partitions of the data simultaneously. Model parallelism, also known as network parallelism, is a strategy that divides the workload according to the neurons in each layer into multiple PUs. Here, the data are copied to all PUs, and different parts of the DNN are computed on different PUs. Finally, in deep learning, layer pipelining can refer to overlapping computations, i.e., between one layer and the next, or partitioning the DNN according to depth and assigning layers to specific processors. Layer pipelining can be viewed as a form of data parallelism, since data are processed through the network in parallel, but also as model parallelism, since the DNN structure determines the pipeline length [11]. Parallelization is considered a promising solution to overcome the acceleration challenges of large DNN models. However, this technique increases resource usage and power, making it more challenging to apply large models in time-critical or resource-limited systems.

Abdelouahab et al. [2] proposed a **Direct Hardware Mapping (DHM)** approach for FPGA-based CNN acceleration by fully exploiting the CNN inter-neuron parallelism (Figure 10). The proposed approach maps the CNN as a dataflow network and uses a data-driven execution scheme to trigger the network execution upon the availability of inputs. The entire graph is mapped onto the FPGA to leverage CNN parallelism and avoid off-chip accesses. Their design processes streams of feature maps as they arrive and fully pipelines the execution. DHM achieved a maximum throughput of 437 GOP/s with the CIFAR-10 CNN model on the Zynq XC706 FPGA with roughly 80% LUT usage. Furthermore, T-DLA [25] leverages the DSPs of the FPGA to operate in **Single Instruction Multiple Data (SIMD)** mode for addition operations. By splitting the input/output into smaller

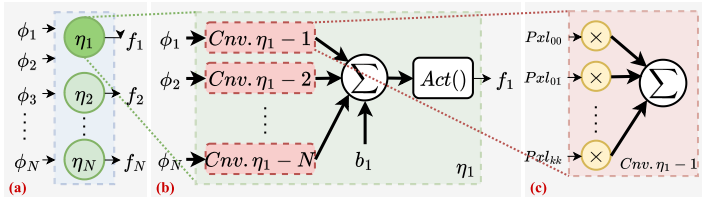

Fig. 10. The three levels of DHM used on CNN: (a) in the convolution layers, (b) in the neurons (ηx), (c) in the convolution engines ($cnv.\eta_x$-n), adapted from Reference [2].

data segments, the DSP can operate independently on all segments without carry propagation between them. To take advantage of a TNN-specific training solution that restricts features to less than 12 bits, the authors split the original 48-bit DSP input into four independent accumulation channels, allowing a single DSP to complete addition operations for eight pieces of input data and provide four outputs. SIMD mode enables the DSP to provide output results in every clock cycle.

4.2.3 Loop optimizations. Generally, current FPGAs' on-chip memory capacity is not sufficiently large to store all DNN layers' weights and intermediate feature maps. Therefore, FPGA-based accelerators require the use of external DRAMs to store this data. DRAM accesses are very costly in terms of latency and energy, and data caches have to be implemented using local registers and on-chip buffers. The challenge is to configure a data path that ensures the maximum reusability of all the data transferred from the DRAM. This goal can be accomplished by using loop optimization methods such as loop tiling and loop unrolling.

Zhang et al. [178] use the roofline model to present an analytical approach for optimizing CNN performance on FPGAs. They utilize *polyhedral-based data dependence analysis* for seeking suitable variants of a CNN using different loop tile sizes and loop schedules. They use polyhedral-based optimization to reduce loop-carried dependences for permuting parallel loop levels to the innermost levels, since loop-carried dependence prevents full pipelining of the loops. After this, loop pipelining is applied. Loop tiling is used for loops with large loop bounds, as tiling loops with small loop bounds do not increase performance. The accelerator was tested using a custom CNN on the VC707 board and achieved a throughput of 61.62 GOP/s at 100 MHz and energy efficiency of 3.3 GOP/s/W while scoring a high accuracy. To evaluate different CNN accelerators' configurations on FPGAs, Rahman et al.[127] perform **Design Space Exploration (DSE)**. DSE was used in several dimensions, such as processing array shapes and sizes, loop tiling/unrolling/reordering, and so on. Under the constraints of DSP, RAM, and off-chip memory bandwidth resources, they seek to minimize the execution cycle and minimize bandwidth consumption for equal cycles. They found that the best MAC array shape is strongly influenced by the CNN and, to some degree, by the FPGA.

4.2.4 Data Reuse. Data reuse aims at minimizing memory access. The core idea of data reuse is to deploy an on-chip buffer to store data that will be used repeatedly. Moini et al.[109] show that convolution operations can be boosted by maximizing data reuse. This approach relies on pixels that are simultaneously computed at the same spatial position of various OFMs. Therefore, all pixels of IFMs and kernel data are accessed only once and stored in an on-chip SRAM until their reuse is depleted. This data-reuse approach was implemented using fixed-point arithmetic. Peemen et al. [122] propose a versatile off-chip memory hierarchy approach to create a memory-centric accelerator template for several CNN models. The memory-centric accelerator utilizes a SIMD cluster of MAC PEs with versatile reuse buffers to accelerate the convolutional layer. The proposed data reuse approach exploits the intricate access patterns to reduce off-chip

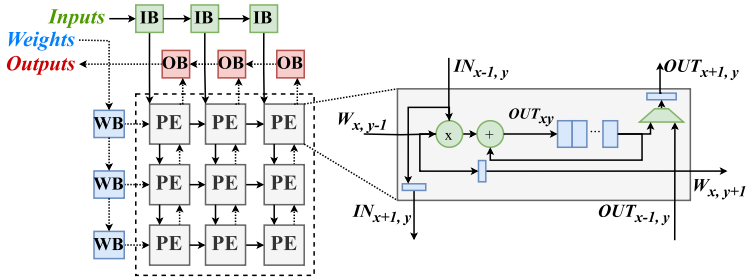

Fig. 11. Systolic array architecture for CNN and the PE structure as proposed by Wei et al. [164].

memory communication, minimizing bandwidth requirements. Using loop transformation and BRAM-based on-chip buffers, the memory-centric accelerator optimizes the efficiency of on-chip memories for better data locality [13]. The results show that the memory-centric accelerator is 11× faster than the standard implementation of similar FPGAs resources and more energy-efficient due to the reduced memory access.

4.2.5 Systolic arrays. Systolic arrays are hardware structures designed as an array of processing elements *PEs* that can be programmed to perform a specific operation, such as matrix-matrix multiplication. Regularity, reconfigurability, and scalability are some of the features of the systolic design. In systolic arrays, once data are loaded from memory, they can be used effectively at each *PE* of the array while being passed from $PE - to - PE$ along the array, thus avoiding the classic memory access bottleneck commonly faced by Von Neumann machines.

Conventional methods employ loop unrolling on multiple for-loops to enable a high degree of parallelization and perform memory partitioning to provide multiple data to different DSP components. These designs may have difficulty meeting the timing constraints for a high clock frequency or even passing the place and route phase. The timing issue is mainly caused by the large-scale data reuse between DSP blocks that introduces a large fan-out. Direct interconnects become long wires when the DSP blocks are distributed throughout the FPGA device, increasing the need for large multiplexes for the output data collection. Wei et al. [164] propose a 2D systolic array design that eliminates multiplexes and reduces the length of interconnect wires by shifting input/output data between neighboring **processing elements (PEs)** both horizontally and vertically. According to the authors, systolic arrays are highly adaptable to different DNNs and can achieve high parallelism and clock frequencies due to their suitability for place and route. Figure 11 shows the architectures of a PE with index (x, y) and buffer structures (IB, OB, and WB). Input feature map data are pipelined across the IB chain while each IB selectively stores data belonging to the corresponding PE column. Similar structures are used for weight WBs and output feature map OBs to fully pipeline the entire systolic array.

In Reference [62], a dedicated systolic array-based LSTM engine was proposed to efficiently solve large matrix multiplication problems in LSTM. The systolic array employs the 12-bit fixed point format to perform matrix multiplication and passes the results to the *activation* and *element-wise computation* units. The LSTM engine exhibits a throughput and energy-efficiency improvement of up to 8.8× and 16.9× on the Xilinx XCKU115 compared to CPU and GPU. In Reference [23], a systolic array architecture for accelerating the multi-head attention and the feed-forward network parts of Transformers is presented. Each PE array is mapped to individual DSPs, leveraging fully from their interconnects at up to 588 MHz on the Xilinx ZCU102 FPGA. The proposed design achieves 1.8× higher frequency than previous designs [87, 102, 159] while using fewer LUT and BRAM resources. Authors in Reference [33] presented two proposals for

Stacked Sparse Autoencoder (SSAE) acceleration on FPGA. Both proposals employ a systolic array approach for higher scalability and faster processing. In general, the presented proposal achieved high throughput, with proposal 2 employing LUTs to store the weights on-chip, which reduced resource usage and increased throughput compared to proposal 1. In contrast, proposal 1 allows new weights to be loaded to the design through weight streams, making it suitable for different problems using the same hardware. Experimental results indicate that the proposed design can achieve speeds 20× faster and a far higher energy efficiency than presented in prior works [58, 156, 178, 179].

4.3 Implementation Optimization

4.3.1 Design Space Exploration. DSE refers to the search for hyper-parameters that satisfy the design constraints imposed on the accelerator design regarding throughput, resource usage, latency, and so on. This exploration process is naturally complex, as the search may involve designs generated by applying code transformations and optimizations at various levels of abstraction, the selection of specific values of parameters, or even selecting algorithmic alternatives. Therefore, the selection of alternative design options and the corresponding design generation and evaluation should be automated due to the magnitude of the search space.

Although exhaustive search methods such as heuristics, dynamic programming, or branch-and-bound can be effective in some cases, they are probably feasible for some design cases. DSE techniques should produce designs that meet objectives within tenable timeframes [21]. Various algorithmic approaches, including stochastic optimization techniques such as random search, evolutionary algorithms, swarm algorithms, and ML, are used in DSE [35]. Each design point can be evaluated empirically using measurements or simulations or with performance models, potentially analytical, to obtain necessary information. The desired designs can be categorized as a single objective, possibly with multiple criteria, or multi-objective, depending on the optimization criteria [119].

Shan et al. [133] propose a two-step heuristic solution to optimize DNN accelerator mapping on multiple FPGAs. The heuristic algorithm is designed to handle the limitations of modern FPGAs regarding memory bandwidth and resources while optimizing the assignment of **Compute Units (CUs)** on multiple FPGAs and minimizing the **initiation interval (II)**. The experimental results show a 40% reduction in II and up to a 20% reduction in resource usage compared to prior works. In addition, the heuristic algorithm is 2 to 3× faster than the previously used solver and produces comparable results. In Reference [96], the E-RNN framework was introduced to improve performance and energy efficiency under the accuracy requirement for speech recognition. This framework begins with determining the RNN model to reduce computation and storage while maintaining accuracy. Then, it focuses on the hardware implementation of the RNN model, including processing element design, quantization, and activation implementation. Experimental results on actual FPGA reveal that E-RNN achieves a maximum energy-efficiency improvement of 37.4× compared to ESE [60] and more than 2× compared to C-LSTM [160]

4.3.2 Synchronous Dataflow. Due to DNN models applying fixed operations to the input data at predefined stages of the execution pipeline, they can be represented as computational graphs. This allows **Synchronous Dataflow (SDF)** to be used for the hardware mapping of a DNN. SDF is a data-flow-driven computing paradigm for designing parallel hardware and software components. In this model, a computing system is represented as a directed graph, called an **SDF graph (SDFG)**, where nodes and arcs represent computations and data streams, respectively. The underlying concept of SDF is that once a token (input data) is available on all input arcs, the connected nodes fire. SDF offers the potential to generate static execution schedules.

The SDF model can then be used to simplify the mapping of DNN graphs to FPGAs. Here, the SDFG is constructed by mapping each layer of the DNN to a hardware building block representing the nodes of the SDFG. The graph is subsequently modeled as a topology matrix, allowing the DSE to be expressed as a mathematical optimization problem. Venieris et al. [152, 154] proposed fpgaConvNet, a dedicated design flow for generating high-throughput hardware designs that meet the performance requirements of high-throughput applications. fpgaConvNet exploits batching and the reconfigurability of FPGAs to generate an optimized high-throughput architecture for CNNs on the target platform. Furthermore, it employs an SDF model to represent both CNNs workloads and hardware architectures. Under this scheme, the design space is explored through a set of algebraic transformations that adjust the performance-resource attributes of the hardware design.

4.3.3 FPGA-tailored DNN Architectures. DNNs are an ideal candidate for custom hardware acceleration owing to their intrinsic parallelism. Researchers increasingly resort to low-precision data types to achieve higher performance within resource and power constraints. In this context, BNN compact structures offer a highly parallelizable solution. Nevertheless, when deployed on FPGAs, their simplicity often results in underutilizing the target platform's extensive computational and routing resources.

LUT-based DNN inference accelerators have achieved remarkable performance when deployed on FPGAs. NullaNet [113] and LogicNets [149] are frameworks for designing neural network topologies that can be directly implemented on an FPGA for highly efficient and parallel processing. They replace the expensive DNN operations with Boolean logic expressions that can be mapped to native LUTs on FPGA devices. Both methodologies involve quantization-aware training, fan-in-constrained pruning for neurons, truth table representation, and multi-level logic minimization and retiming. However, they were designed with small-scale classification tasks in mind, for which they reached latency in the tens of nanoseconds and throughput in the hundreds of millions of samples per second.

LUTNet [157, 158] is a hardware-software framework that leverages the native LUTs in FPGAs as inference operators to build area-efficient binary neural network accelerators. The architecture features K-LUTs as inference operators, where each K-LUT can perform an arbitrary Boolean operation on its K binary inputs. LUTNet targets low-precision applications requiring high throughput and low latency dataflow architectures. For the final mapping, XNOR gates, initially used for training, are converted into K-LUTs, and the network is retrained using the K-LUT representation. Experimental results show that LUTNet achieves almost twice the area efficiency and 6.66× the energy efficiency compared to the pruned and unrolled ReBNet [49] architecture due to its efficient use of soft logic. However, the limitation of LUTNet is its scalability. To address this limitation, the tradeoff between area, energy, and throughput can be explored and controlled via the tiling technique.

4.4 Implementation Results and Comparison

This section presents a comparative analysis of over 120 FPGA-based DL inference accelerators,[1] aiming to evaluate the optimization techniques outlined in Sections 4.1, 4.2, and 4.3. Given the diverse techniques, FPGA chips, models, and architectures used, we focus on designs with full system implementations and reported throughput, power, and energy-efficiency measures to ensure a fair comparison.

Among the designs, 1- to 2-bit-based designs demonstrate outstanding speed and energy efficiency, highlighting the potential of using extremely low bit-widths for high-performance design.

[1]The complete list of analyzed FPGA-based accelerators can be found under this link: FPGA-based-DNN-Accels

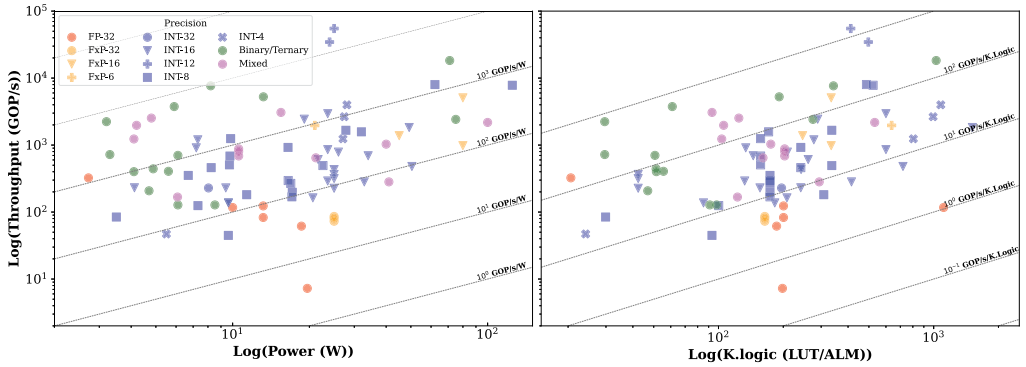

Fig. 12. Energy efficiency (left side) and logic density (right side) plots of FPGA-based DL inference accelerators.

However, binarized and ternarized models suffer from notable accuracy loss and, in some cases, hardware underutilization. Further research aims to harness BNN capabilities for FPGA-tailored DNN architectures, optimizing throughput and energy efficiency by leveraging FPGA native LUTs while minimizing accuracy loss. Trading speed for accuracy can be achieved with higher bit widths, such as INT-4 or INT-6. Nearly 63% of the surveyed accelerators utilize integer datatypes for representing weights and activations. Specifically, INT-16 and INT-8 are favored due to their excellent tradeoff between accuracy, energy efficiency, and computational density, as illustrated in Figure 12. The advantage of INT-16 over 32-bit floating-point designs is evident, except for Reference [180], where hard-core floating-point DSPs are employed, emphasizing the importance of fully utilizing on-chip DSPs to some extent.

Pruning and tensor decomposition reduce DNNs' computational costs. Combined with quantization, pruning significantly reduces model size, memory, and computational requirements. However, it may need retraining and fine-tuning to limit accuracy loss. Structured pruning is more suited for accelerating RNN [19] and Transformers [123, 124] models without adding irregularities to computations. Channel-level pruning offers potential improvement with no additional data structures or training overhead. Tensor decomposition methods such as CPD and BMD reduce computation costs in DNNs with minimal accuracy loss. However, implementing them in large DNNs poses challenges due to the exponential increase in training time, which can be addressed by incorporating hyper-parameters during training [52].

Batching and data reuse are architectural optimizations aiming to reduce memory access by simultaneously processing a batch of data or buffering frequently used data on-chip. This results in reduced latency and improved power efficiency. However, in the fpgaConvnet [152, 153], the authors did not employ batching as discussed in Section 4.2.5. Instead, they used it only for off-chip memory data loading or write-back. This approach limits the platform batch size to a maximum of 254 inputs, the burst limit of the Video DMA IP, making it a performance bottleneck in their design.

Additionally, parallelization and loop optimization can further enhance design efficiency, but they come at the expense of increased resource utilization and power consumption. The usage of these methods depends on the DNN model's size, topology, and the targeted FPGA device. Moreover, systolic arrays can be employed with SIMD units to achieve high performance but usually suffer from bottlenecks during data transfer to and from external memory [155], which can be mitigated by adding more buffers for weights and input data. While systolic arrays offer high parallelism and are adaptable to various DNNs due to their ability to achieve high clock

frequencies and suitability for place and route, their flexibility results in reduced efficiency and varying throughputs for different DNNs [56].

FPGAs are a promising medium for implementing DNNs due to their reconfigurability and efficiency. However, finding optimal solutions is challenging, given the vast joint search space for DNN topologies, model parameters, and hardware constraints. An automatic framework that explores such a vast space and finds optimal solutions is highly desirable. In general, DSE methods that allow for finer customization offer better results. For example, fpgaConvNet's [153] analytical methodology outperformed DnnWeaver's [134] slightly more restricted design space, which, in turn, outperformed DeepBurning's [163] heuristic mapping. Additionally, SysArrayAccel's [164] detailed design space exploration method enabled the traversal of a larger design space than DnnWeaver, resulting in higher-performing designs.

The representation of computation using a **dataflow graph (DFG)** is a common practice, with nodes representing operators and edges representing data dependencies. Given the graph-like structures of DNNs, it is natural to represent DL models as DFGs [1]. While previous approaches, such as loop analysis, have gained popularity due to their performance, some contributions attempt to convert DFGs to FPGA implementations and processing schedules using graph-aware templates. With DFG, the processing rates of all blocks in the system are known *a priori*. Therefore, a fixed schedule is generated to drive the datapath. DFGs often achieve high performance and energy efficiency, since they generate custom data paths and computing engines for a given DNN model. Hence, they suffer from poor scalability, since a dataflow-based accelerator for one model cannot be used to accelerate another [155].

5 DNN ACCELERATION CRITERIA AND FPGA-BENCHMARKING

Specific acceleration criteria must be considered when designing an FPGA-based DNN inference accelerator. This section introduces standard acceleration criteria such as inference throughput and energy efficiency, then highlights one more specific to FPGAs: computational density. These criteria allow the comparison of hardware designs among various FPGA technologies. Furthermore, experimental results will be presented to show the acceleration techniques' effect on the mentioned criteria.

5.1 Inference Throughput

The first acceleration criterion is throughput. In this context, DNN model simplification techniques and parallelization are the most prominent acceleration techniques for increasing the DNN inferences' peak performance. While parallelization allows operations to be executed simultaneously, quantization, as one of DNNs' model simplification techniques, reduces the memory requirements of the model parameters by representing them with fewer bits. Consequently, this approach can allow on-chip storage of the parameters, eliminating the time needed to load them from external memory. Pruning is another simplification technique that reduces computational complexity by setting individual parameters to zero.

To back our claim on how the mentioned acceleration techniques can increase the inference throughput of FPGA-based DNN accelerators, we perform several experiments to accelerate a ResNet-2 model trained on CIFAR-10 and an LSTM model (comprising 2 LSTM layers and 3 Dense Layers) for airline passengers prediction [80] using hls4ml on the ZCU104 FPGA. First, we generate different accelerators with various pruning, quantization, and reuse factor configurations. Here, the reuse factor changes the parallelism of the implementation by affecting the degree of loop unrolling. It should be noted that hls4ml is chosen as an exemplary toolflow to perform our benchmarking and generate DNN accelerators using different optimization techniques to back our claims regarding the impact of selected optimizations.

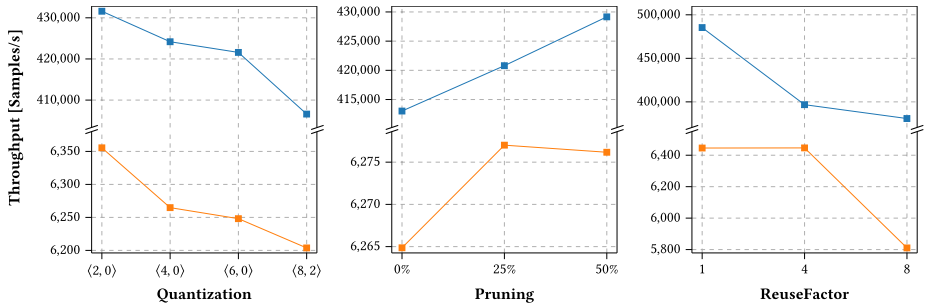

Fig. 13. Quantization, Pruning, and Parallelization effects on inference throughput for LSTM (■) and ResNet-2 (■). For quantization, the fixed-point identifier $\langle W, I \rangle$ is used where W is the word length, and I is the number of bits used for the integer part, both in bits.

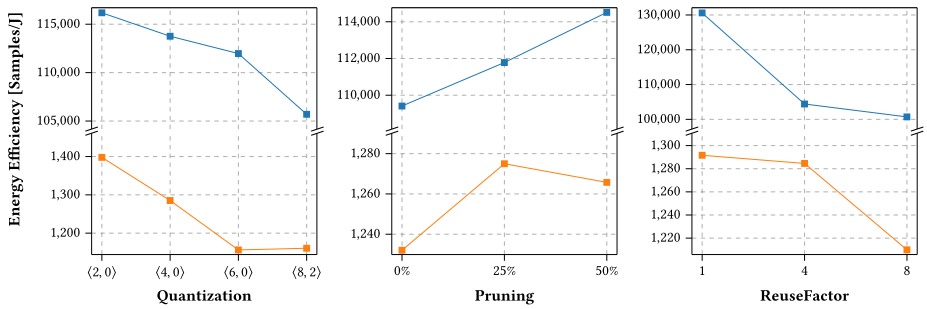

Fig. 14. Quantization, Pruning, and Parallelization effects on energy efficiency for LSTM (■) and ResNet-2 (■).

Figure 13 shows the pruning, quantization, and reuse factor effects on the throughput of our proposed ResNet- and LSTM-based accelerators. Quantization affects the inference throughput in that it decreases with the increased bit-width representation of both model parameters. The reuse factor is a parameter that determines the number of times each multiplier is used to compute layers of neuron values. It thus describes the level of parallelization in each layer. Here, a reuse factor of 1 means that each multiplier is used for only one multiplication, resulting in full parallelization and higher throughput. Consequently, increasing this parameter leads to a decrease in throughput. The reuse factor of 8 indicates that one multiplier executes eight consecutive multiplications, leading to a drop in throughput. Moreover, increasing the pruning ratio improves the inference speed in general, especially in the case of the LSTM model, which could be explained by the fact that the pruning helped reduce the data dependencies and computational complexity for such a model.

5.2 Energy Efficiency

The second acceleration criterion is energy efficiency. Generally, energy efficiency can be expressed as the inference throughput over energy consumption. For instance, the energy efficiency of a DNN model for image classification is expressed by $Frames/sec/Watt$ or simply $Frames/Joule$. Similar to the first criterion, quantization, pruning, and parallelization are employed for our ResNet-2 and LSTM-based accelerator, and their impact on the energy efficiency of an FPGA-based DNN inference accelerator is analyzed.

Figure 14 shows the effect of such techniques on energy efficiency. More precisely, quantization helps achieve better energy efficiency by reducing the model parameters' storage size, resulting in

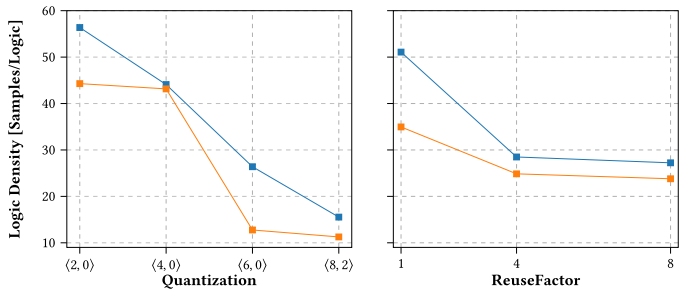

Fig. 15. Quantization and Parallelization effects on computational density for LSTM (■ expressed in Samples/LUT) and ResNet-2 (■ expressed in Samples/DSP).

less memory resource usage, such as BRAM and UltraRAM cells. This is more significant for the LSTM model, since it features higher memory requirements than the ResNet-2 model. Furthermore, the reduced bit width representation of the parameters helps reduce the computational complexity. For instance, in the case of BNNs, the matrix-vector multiplication arithmetics are replaced with simple *XNOR* operations, resulting in reduced LUT and DSP usage and consequently higher throughput with less power consumption [114]. Pruning also achieves better energy efficiency by reducing the computation and the memory overhead. Though its effect is not as significant as quantization for the ResNet-2, the LSTM-based accelerator clearly benefits from the reduced computation complexity and data dependency. However, parallelization increases resource utilization to enable more parallel operations and improve the throughput. However, if routing becomes too complex for highly parallel designs, then logic overhead may be introduced, and the improvement in throughput may be insufficient to compensate for increased energy requirements.

5.3 Computational Density

Computational density is a metric used in FPGA design, referring to the ratio of computations performed by a particular design over the number of resources utilized. In other words, this criterion indicates whether the hardware design suffers from resource underutilization or not. For instance, when two different accelerators deliver the same inference throughput, the one with the lowest DSP usage is considered better in terms of computational density. Nevertheless, in some cases, some designs rely on LUTs only, such as BNN and TNN, and the DSP usage becomes irrelevant. Therefore, the computational density becomes a metric reflecting the LUT density. Generally, the computational density is expressed as $Throughput/\#DSP$ or $Throughput/\#LUT$. Figure 15 demonstrates the effect of quantization and parallelization on the logic density. It is clear that using reduced bit-width representations increases the logic density in general, and we can almost note the same behavior for design relying on DSP- or LUT-based multipliers. Highly parallel designs are also characterized by a higher logic density. We can see that increasing the reuse factor leads to a decreased logic density for both models, resulting in resource underutilization. We noticed that pruning did not significantly affect logic density, as we can see a variation of a maximum 1% on average. However, it should be mentioned that the input data, model topology, and FPGA technology also impact this metric.

6 CONCLUSION

This survey article aims to provide a deep insight into FPGA-based DL accelerator design. We first investigated the state-of-the-art and challenges of FPGA-based DL inference accelerators. We detailed recent DNN acceleration strategies, including (1) various model simplification approaches

such as quantization, pruning, and tensor decomposition, (2) architectural optimizations such as batching, parallelization, and systolic arrays, and (3) optimization approaches, namely, Design space exploration, Synchronous Dataflow, and FPGA-tailored DNN architecture. We evaluated many of these strategies in practice based on proposed acceleration criteria such as inference throughput, energy efficiency, and computational density.

As most of the literature focuses on CNN due to its colossal emergence in many domains, we performed a practical experiment in DNN-based accelerators. Particularly, we constructed two DNN models: ResNet-2 and LSTM, as examples of unconventional CNNs and RNNs. Then, we implemented different configurations of ResNet-2 and LSTM-based accelerators, such as pruning, quantization, and parallelization, and we benchmarked and compared these accelerators based on the proposed acceleration criteria. Our results indicate that quantization can be perceived as a pragmatic optimization approach for FPGA-based DL accelerators, exhibiting higher throughput and lower energy consumption than other optimization techniques.

Further, we showed and reviewed several FPGA- based accelerators and their acceleration strategies. According to our review and observations, we expect to see more focus on accelerating Transformers in the near future, and we anticipate FPGA-tailored DNN architectures to become more adaptable to even larger models, not just restricted to small classification problems. As a conclusion of this survey, our observations and benchmarking can serve as a guideline for building FPGA-based DL accelerators.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their supportive comments.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, 265–283. Retrieved from https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, Cedric Bourrasset, and François Berry. 2017. Tactics to directly map CNN graphs on embedded FPGAs. *IEEE Embed. Syst. Lett.* 9, 4 (2017), 113–116.

[3] Adil Al-Azzawi, Anes Ouadou, Highsmith Max, Ye Duan, John J. Tanner, and Jianlin Cheng. 2020. DeepCryoPicker: Fully automated deep neural network for single protein particle picking in cryo-EM. *BMC Bioinform.* 21, 1 (2020), 1–38.

[4] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot. 2017. Ternary neural networks for resource-efficient AI applications. In *International Joint Conference on Neural Networks (IJCNN'17)*. IEEE, 2547–2554. DOI: https://doi.org/10.1109/IJCNN.2017.7966166

[5] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. 2021. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* 8, 1 (2021), 1–74.

[6] Christiaan Baaij, Matthijs Kooijman, Jan Kuper, Arjan Boeijink, and Marco Gerards. 2010. C? ash: Structural descriptions of synchronous hardware using Haskell. In *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. IEEE, 714–721.

[7] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: Constructing hardware in a Scala embedded language. In *49th Annual Design Automation Conference*. 1216–1225.

[8] Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Adv. Neural Inf. Process. Syst.* 32 (2019).

[9] Özlem Batur Dinler and Nizamettin Aydin. 2020. An optimal feature parameter set based on gated recurrent unit recurrent neural networks for speech segment detection. *Appl. Sci.* 10, 4 (2020), 1273.

[10] Peter Bellows and Brad Hutchings. 1998. JHDL-an HDL for reconfigurable systems. In *IEEE Symposium on FPGAs for Custom Computing Machines*. IEEE, 175–184.
[11] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Comput. Surv.* 52, 4 (2019), 1–43.
[12] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 8 (2013), 1798–1828.
[13] Aleksandar Beric, Jef van Meerbergen, Gerard de Haan, and Ramanathan Sethuraman. 2008. Memory-centric video processing. *IEEE Trans. Circ. Syst. Vid. Technol.* 18, 4 (2008), 439–452.
[14] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 129–146. Retrieved from https://proceedings.mlsys.org/paper_files/paper/2020/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf
[15] Michaela Blott, Lisa Halder, Miriam Leeser, and Linda Doyle. 2019. QuTiBench: Benchmarking neural networks on heterogeneous hardware. *ACM J. Emerg. Technol. Comput. Syst.* 15, 4 (2019), 1–38.
[16] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'Brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. Reconfig. Technol. Syst.* 11, 3 (2018), 1–23.
[17] Andrew Boutros and Vaughn Betz. 2021. FPGA architecture: Principles and progression. *IEEE Circ. Syst. Mag.* 21, 2 (2021), 4–29.
[18] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep learning with low precision by half-wave Gaussian quantization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*.
[19] Shijie Cao, Chen Zhang, Zhuliang Yao, Wencong Xiao, Lanshun Nie, Dechen Zhan, Yunxin Liu, Ming Wu, and Lintao Zhang. 2019. Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19)*. Association for Computing Machinery, New York, NY, 63–72. DOI: https://doi.org/10.1145/3289602.3293898
[20] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Muhammad Shafique, Guido Masera, and Maurizio Martina. 2020. An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. *Fut. Internet* 12, 7 (2020), 113.
[21] João M. P. Cardoso, José Gabriel F. Coutinho, and Pedro C. Diniz. 2017. Additional topics. In *Embedded Computing for High Performance*, JoÃčo M. P. Cardoso, José Gabriel F. Coutinho, and Pedro C. Diniz (Eds.). Morgan Kaufmann, Boston, 255–280. DOI: https://doi.org/10.1016/B978-0-12-804189-5.00008-9
[22] W. Chen, D. Xie, Y. Zhang, and S. Pu. 2019. All you need is a few shifts: Designing efficient convolutional neural networks for image classification. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'19)*. 7234–7243. DOI: https://doi.org/10.1109/CVPR.2019.00741
[23] Yonghao Chen, Tianrui Li, Xiaojie Chen, Zhigang Cai, and Tao Su. 2023. High-frequency systolic array-based transformer accelerator on field programmable gate arrays. *Electronics* 12, 4 (2023), 822.
[24] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. 2020. A survey of accelerator architectures for deep neural networks. *Engineering* 6, 3 (2020), 264–274. DOI: https://doi.org/10.1016/j.eng.2020.01.007
[25] Yao Chen, Kai Zhang, Cheng Gong, Cong Hao, Xiaofan Zhang, Tao Li, and Deming Chen. 2019. T-DLA: An open-source deep learning accelerator for ternarized DNN models on embedded FPGA. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI'19)*. IEEE, 13–18.
[26] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-state Circ.* 52, 1 (2016), 127–138.
[27] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Select. Topics Circ. Syst.* 9, 2 (2019), 292–308.
[28] Francois Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*.
[29] Claudionor N. Coelho Jr, Aki Kuusela, Shan Li, Hao Zhuang, Jennifer Ngadiuba, Thea Klaeboe Aarrestad, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers. 2021. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nat. Mach. Intell.* 3, 8 (2021), 675–686.
[30] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv preprint arXiv:1602.02830* (2016).
[31] Philippe Coussy, Daniel D. Gajski, Michael Meredith, and Andres Takach. 2009. An introduction to high-level synthesis. *IEEE Des. Test. Comput.* 26, 4 (2009), 8–17.
[32] Philippe Coussy and Adam Morawiec. 2008. *High-level Synthesis: From Algorithm to Digital Circuit*. Springer Science & Business Media.

[33] Maria G. F. Coutinho, Matheus F. Torquato, and Marcelo A. C. Fernandes. 2019. Deep neural network hardware implementation based on stacked sparse autoencoder. *IEEE Access* 7 (2019), 40674–40694.
[34] W. Dai and D. Berleant. 2019. Benchmarking contemporary deep learning hardware and frameworks: A survey of qualitative metrics. In *IEEE 1st International Conference on Cognitive Machine Intelligence (CogMI'19)*. 148–155. DOI: https://doi.org/10.1109/CogMI48466.2019.00029
[35] Kalyanmoy Deb. 2011. Multi-objective optimisation using evolutionary algorithms: An introduction. In *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*. Springer, 3–34.
[36] Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. 2018. GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Netw.* 100 (2018), 49–58.
[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://aclanthology.org/N19-1423
[38] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *42nd Annual International Symposium on Computer Architecture*. 92–104.
[39] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran, and Z. Wu. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *J. Instrument.* (July 2018). DOI: https://doi.org/10.1088/1748-0221/13/07/p07027
[40] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural acceleration for general-purpose approximate programs. In *45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 449–460.
[41] Weili Fang, Peter E. D. Love, Hanbin Luo, and Lieyun Ding. 2020. Computer vision for behaviour-based safety in construction: A review and future directions. *Adv. Eng. Inform.* 43 (2020), 100980.
[42] Xin Feng, Youni Jiang, Xuejiao Yang, Ming Du, and Xin Li. 2019. Computer vision algorithms and hardware implementations: A survey. *Integration* 69 (2019), 309–320.
[43] Martin Fowler. 2010. *Domain-specific Languages*. Pearson Education.
[44] Cheng Fu, Shilin Zhu, Hao Su, Ching-En Lee, and Jishen Zhao. 2018. Towards fast and energy-efficient binarized neural network inference on FPGA. *arXiv preprint arXiv:1810.02068* (2018).
[45] Daniel D. Gajski, Nikil D. Dutt, Allen C. H. Wu, and Steve Y. L. Lin. 2012. *High-level Synthesis: Introduction to Chip and System Design*. Springer Science & Business Media.
[46] Chang Gao, Daniel Neil, Enea Ceolini, Shih-Chii Liu, and Tobi Delbruck. 2018. DeltaRNN: A power-efficient recurrent neural network accelerator. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. 21–30.
[47] Chang Gao, Junkun Yan, Shenghua Zhou, Pramod K. Varshney, and Hongwei Liu. 2019. Long short-term memory-based deep recurrent neural networks for target tracking. *Inf. Sci.* 502 (2019), 279–296. DOI: https://doi.org/10.1016/j.ins.2019.06.039
[48] Tong Geng, Ang Li, Tianqi Wang, Chunshu Wu, Yanfei Li, Runbin Shi, Wei Wu, and Martin Herbordt. 2020. O3BNN-R: An out-of-order architecture for high-performance and regularized BNN inference. *IEEE Trans. Parallel Distrib. Syst.* 32, 1 (2020), 199–213.
[49] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. 2018. ReBNet: Residual binarized neural network. In *IEEE 26th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'18)*. IEEE, 57–64.
[50] Seyed Abolfazl Ghasemzadeh, Erfan Bank Tavakoli, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2021. BRDS: An FPGA-based LSTM accelerator with row-balanced dual-ratio sparsification. *arXiv preprint arXiv:2101.02667* (2021).
[51] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *13th International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 249–256.
[52] Abhinav Goel, Caleb Tung, Yung-Hsiang Lu, and George K. Thiruvathukal. 2020. A survey of methods for low-power deep learning and computer vision. In *IEEE 6th World Forum on Internet of Things (WF-IoT'20)*. IEEE, 1–6.
[53] Lovedeep Gondara. 2016. Medical image denoising using convolutional denoising autoencoders. In *IEEE 16th International Conference on Data Mining Workshops (ICDMW'16)*. IEEE, 241–246.
[54] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen 2018. Recent advances in convolutional neural networks. *Pattern Recognition* 77 (2018), 354–377. https://www.sciencedirect.com/science/article/pii/S0031320317304120
[55] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. 2017. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS

hybrid templates. In *IEEE 25th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'17)*. IEEE, 152–159.

[56] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. 2017. Angel-eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 37, 1 (2017), 35–47.

[57] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2019. [DL] A survey of FPGA-based neural network inference accelerators. *ACM Trans. Reconfig. Technol. Syst.* 12, 1 (2019), 1–26.

[58] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*. PMLR, 1737–1746.

[59] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. 2020. GhostNet: More features from cheap operations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1580–1589.

[60] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William Bill J. Dally. 2017. ESE: Efficient speech recognition engine with sparse LSTM on FPGA. *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17, Monterey, California, USA)*, Association for Computing Machinery, New York, NY, 75–84. https://doi.org/10.1145/3020078.3021745

[61] Yue Han, Qiu-Hua Lin, Li-Dan Kuang, Xiao-Feng Gong, Fengyu Cong, Yu-Ping Wang, and Vince D. Calhoun. 2021. Low-rank Tucker-2 model for multi-subject fMRI data decomposition with spatial sparsity constraint. *IEEE Trans. Med. Imag.* 41, 3 (2021), 667–679.

[62] Dazhong He, Junhua He, Jun Liu, Jie Yang, Qing Yan, and Yang Yang. 2021. An FPGA-based LSTM acceleration engine for deep learning frameworks. *Electronics* 10, 6 (2021), 681.

[63] D. He, Z. Wang, and J. Liu. 2018. A survey to predict the trend of AI-able server evolution in the cloud. *IEEE Access* 6 (2018), 10591–10602. DOI: https://doi.org/10.1109/ACCESS.2018.2801293

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[65] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision*. 1389–1397.

[66] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. 2021. Recurrent neural networks for time series forecasting: Current status and future directions. *Int. J. Forecast.* 37, 1 (2021), 388–427.

[67] Frank L. Hitchcock. 1927. The expression of a tensor or a polyadic as a sum of products. *J. Math. Phys.* 6, 1-4 (1927), 164–189.

[68] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV'19)*.

[69] 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).

[70] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. 2017. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*. 2261–2269. DOI: https://doi.org/10.1109/CVPR.2017.243

[71] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *Adv. Neural Inf. Process. Syst.* 29 (2016).

[72] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18, 1 (2017), 6869–6898.

[73] Lucas Y. W. Hui and Alexander Binder. 2019. BatchNorm decomposition for deep neural network interpretation. In *15th International Work-conference on Artificial Neural Networks: Advances in Computational Intelligence*. Springer, 280–291.

[74] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[75] Google Inc. 2019. *What is the Edge TPU?* Retrieved from https://coral.ai/docs/edgetpu/faq/#what-is-the-edge-tpu

[76] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[77] Abhyuday N. Jagannatha and Hong Yu. 2016. Structured prediction models for RNN based sequence labeling in clinical text. In *Conference on Empirical Methods in Natural Language Processing*, Vol. 2016. NIH Public Access, 856.

[78] Yihan Jiang, Hyeji Kim, Himanshu Asnani, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. 2020. Learn codes: Inventing low-latency codes via recurrent neural networks. *IEEE J. Select. Areas Inf. Theor.* 1, 1 (2020), 207–216.

[79] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter performance analysis of a tensor processing unit. In *44th Annual International Symposium on Computer Architecture (ISCA '17, Toronto, ON, Canada)*, Association for Computing Machinery, New York, NY, 1–12. https://doi.org/10.1145/3079856.3080246

[80] Rakan Nimer. 2017. Air passengers. *Kaggle*. Retrieved from https://www.kaggle.com/datasets/rakannimer/air-passengers

[81] Hamza Khan, Asma Khan, Zainab Khan, Lun Bin Huang, Kun Wang, and Lei He. 2021. NPE: An FPGA-based overlay processor for natural language processing. *CoRR* abs/2104.06535 (2021). https://arxiv.org/abs/2104.06535

[82] Tamara G. Kolda and Brett W. Bader. 2009. Tensor decompositions and applications. *SIAM Rev.* 51, 3 (2009), 455–500.

[83] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. Retrieved from http://www.cs.toronto.edu/kriz/cifar.html

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems Conference*. 1097–1105.

[85] Ping Kuang, Tingsong Ma, Ziwei Chen, and Fan Li. 2019. Image super-resolution with densely connected convolutional networks. *Appl. Intell.* 49, 1 (2019), 125–136.

[86] Griffin Lacey, Graham W. Taylor, and Shawki Areibi. 2016. Deep learning on FPGAs: Past, present, and future. *arXiv preprint arXiv:1602.04283* (2016).

[87] Yi-Hsiang Lai, Hongbo Rong, Size Zheng, Weihao Zhang, Xiuping Cui, Yunshan Jia, Jie Wang, Brendan Sullivan, Zhiru Zhang, Yun Liang, Youhui Zhang, Jason Cong, Nithin George, Jose Alvarez, Christopher Hughes, and Pradeep Dubey. 2020. SuSy: A programming model for productive construction of high-performance systolic arrays on FPGAs. In *39th International Conference on Computer-aided Design (ICCAD'20, Virtual Event, USA)*, Association for Computing Machinery, New York, NY, 1–9. https://doi.org/10.1145/3400302.3415644

[88] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[89] Bingbing Li, Santosh Pandey, Haowen Fang, Yanjun Lyv, Ji Li, Jieyang Chen, Mimi Xie, Lipeng Wan, Hang Liu, and Caiwen Ding. 2020. FTRANS: Energy-efficient acceleration of transformers using FPGA. In *International Symposium on Low Power Electronics and Design (ISLPED'20)*. Association for Computing Machinery, New York, NY, 175–180. DOI: https://doi.org/10.1145/3370748.3406567

[90] 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 *23rd Asia and South Pacific Design Automation Conference (ASP-DAC'18)*. IEEE, 534–539.

[91] Guoqing Li, Meng Zhang, Jiaojie Li, Feng Lv, and Guodong Tong. 2021. Efficient densely connected convolutional neural networks. *Pattern Recog.* 109 (2021), 107610.

[92] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. 2016. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In *26th International Conference on Field Programmable Logic and Applications (FPL'16)*. IEEE, 1–9.

[93] Hsiao-Chi Li, Zong-Yue Deng, and Hsin-Han Chiang. 2020. Lightweight and resource-constrained learning network for face recognition with performance optimization. *Sensors* 20, 21 (2020), 6114.

[94] Wenbin Li and Matthieu Liewig. 2020. A survey of AI accelerators for edge environment. In *World Conference on Information Systems and Technologies*. Springer, 35–44.

[95] Yanbing Li and Miriam Leeser. 2000. HML, a novel hardware description language and its translation to VHDL. *IEEE Tran. Very Large Scale Integ. Syst.* 8, 1 (2000), 1–8.

[96] Zhe Li, Caiwen Ding, Siyue Wang, Wujie Wen, Youwei Zhuo, Chang Liu, Qinru Qiu, Wenyao Xu, Xue Lin, Xuehai Qian, and Yanzhi Wang. 2019. E-RNN: Design optimization for efficient recurrent neural networks in FPGAs. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*. IEEE, 69–80. DOI: 10.1109/HPCA.2019.00028

[97] Zhengjie Li, Yufan Zhang, Jian Wang, and Jinmei Lai. 2020. A survey of FPGA design for AI era. *J. Semiconduct.* 41, 2 (Feb. 2020), 021402. DOI: https://doi.org/10.1088/1674-4926/41/2/021402

[98] Shengwen Liang, Ying Wang, Cheng Liu, Huawei Li, and Xiaowei Li. 2019. InS-DLA: An in-SSD deep learning accelerator for near-data processing. In *29th International Conference on Field Programmable Logic and Applications (FPL'19)*. IEEE, 173–179.

[99] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. DOI: https://doi.org/10.48550/ARXIV.1907.11692

[100] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision*. 2736–2744.

[101] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A ConvNet for the 2020s. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11976–11986.

[102] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. 2020. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. In *IEEE 33rd International System-on-Chip Conference (SOCC'20)*. IEEE, 84–89.

[103] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *European Conference on Computer Vision (ECCV'18)*. 116–131.

[104] Ali Mashtizadeh. 2007. *PHDL: A Python Hardware Design Framework*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[105] Sachin Mehta and Mohammad Rastegari. 2021. MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178* (2021).

[106] Janardan Misra and Indranil Saha. 2010. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing* 74, 1-3 (2010), 239–255.

[107] Sparsh Mittal. 2020. A survey on modeling and improving reliability of DNN algorithms and accelerators. *J. Syst. Archit.* 104 (2020), 101689.

[108] Sparsh Mittal and Jeffrey S. Vetter. 2014. A survey of methods for analyzing and improving GPU energy efficiency. *ACM Comput. Surv.* 47, 2 (2014), 1–23.

[109] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour. 2017. A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications. *IEEE Trans. Circ. Syst. II: Expr. Briefs* 64, 10 (2017), 1217–1221. DOI: https://doi.org/10.1109/TCSII.2017.2690919

[110] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning Convolutional Neural Networks for Resource Efficient Inference. arXiv:1611.06440 [cs.LG].

[111] Aaftab Munshi. 2009. The OpenCL specification. In *IEEE Hot Chips 21 Symposium (HCS'21)*. IEEE, 1–314.

[112] Andriy Myronenko. 2019. 3D MRI brain tumor segmentation using autoencoder regularization. In *4th International Workshop on Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries, Held in Conjunction with MICCAI'18*. Springer, 311–320.

[113] Mahdi Nazemi, Arash Fayyazi, Amirhossein Esmaili, Atharva Khare, Soheil Nazar Shahsavani, and Massoud Pedram. 2021. NullaNet tiny: Ultra-low-latency DNN inference through fixed-function combinational logic. In *IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'21)*. IEEE, 266–267.

[114] E. Nurvitadhi, D. Sheffield, Jaewoong Sim, A. Mishra, G. Venkatesh, and D. Marr. 2016. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In *International Conference on Field-Programmable Technology (FPT'16)*. 77–84. DOI: https://doi.org/10.1109/FPT.2016.7929192

[115] Amos R. Omondi and Jagath Chandana Rajapakse. 2006. *FPGA Implementations of Neural Networks*. Vol. 365. Springer.

[116] Erik Ostrowski and Stefan Kaufmann. 2020. Survey of alternative hardware for neural network computation in the context of computer vision. *researchgate* (2020). https://www.researchgate.net/profile/Erik-Ostrowski/publication/341495230_Survey_of_alternative_hardware_for_Neural_Network_computation_in_the_context_of_Computer_Vision/links/5ec43e8992851c11a877783e/Survey-of-alternative-hardware-for-Neural-Network-computation-in-the-context-of-Computer-Vision.pdf

[117] Adam Page, Ali Jafari, Colin Shea, and Tinoosh Mohsenin. 2017. SPARCNet: A hardware accelerator for efficient deployment of sparse convolutional networks. *J. Emerg. Technol. Comput. Syst.* 13, 3, Article 31 (2017), 32 pages. DOI: https://doi.org/10.1145/3005448

[118] Dimitri Palaz, Mathew Magimai-Doss, and Ronan Collobert. 2019. End-to-end acoustic modeling using convolutional neural networks for HMM-based automatic speech recognition. *Speech Commun.* 108 (2019), 15–32.

[119] Jacopo Panerati and Giovanni Beltrame. 2014. A comparative evaluation of multi-objective exploration algorithms for high-level design. *ACM Trans. Des. Autom. Electron. Syst.* 19, 2 (2014), 1–22.

[120] Alessandro Pappalardo. 2021. *Xilinx/brevitas*. DOI: https://doi.org/10.5281/zenodo.3333552

[121] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2013. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026* (2013).

[122] Maurice Peemen, Arnaud A. A. Setio, Bart Mesman, and Henk Corporaal. 2013. Memory-centric accelerator design for convolutional neural networks. In *IEEE 31st International Conference on Computer Design (ICCD'13)*. IEEE, 13–19.

[123] Hongwu Peng, Shaoyi Huang, Tong Geng, Ang Li, Weiwen Jiang, Hang Liu, Shusen Wang, and Caiwen Ding. 2021. Accelerating transformer-based deep learning models on FPGAs using column balanced block pruning. In *22nd International Symposium on Quality Electronic Design (ISQED'21)*. 142–148. DOI: https://doi.org/10.1109/ISQED51717.2021.9424344

[124] Panjie Qi, Yuhong Song, Hongwu Peng, Shaoyi Huang, Qingfeng Zhuge, and Edwin Hsing-Mean Sha. 2021. Accommodating transformer onto FPGA: Coupling the balanced model compression and FPGA-implementation optimization. In *Great Lakes Symposium on VLSI*. 163–168.

[125] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going deeper with embedded FPGA platform for convolutional neural network. *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16, Monterey, California, USA)*, Association for Computing Machinery, New York, NY, 26–35. https://doi.org/10.1145/2847263.2847265

[126] Zhiqiang Que, Hiroki Nakahara, Hongxiang Fan, Jiuxi Meng, Kuen Hung Tsoi, Xinyu Niu, Eriko Nurvitadhi, and Wayne Luk. 2020. A reconfigurable multithreaded accelerator for recurrent neural networks. In *International Conference on Field-Programmable Technology (ICFPT'20)*. IEEE, 20–28.

[127] Atul Rahman, Sangyun Oh, Jongeun Lee, and Kiyoung Choi. 2017. Design space exploration of FPGA accelerators for convolutional neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*. IEEE, 1147–1152.

[128] Remya Ramakrishnan, Aditya K. V. Dev, A. S. Darshik, Renuka Chinchwadkar, and Madhura Purnaprajna. 2021. Demystifying compression techniques in CNNs: CPU, GPU and FPGA cross-platform analysis. In *34th International Conference on VLSI Design and 20th International Conference on Embedded Systems (VLSID'21)*. IEEE, 240–245.

[129] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner. 2019. Survey and benchmarking of machine learning accelerators. In *IEEE High Performance Extreme Computing Conference (HPEC'19)*. 1–9. DOI: https://doi.org/10.1109/HPEC.2019.8916327

[130] Andres Rodriguez, Eden Segal, Etay Meiri, Evarist Fomenko, Y. Jim Kim, Haihao Shen, and Barukh Ziv. 2018. Lower numerical precision deep learning inference and training. *Intel White Paper* 3 (2018).

[131] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[132] A. Shahid and M. Mushtaq. 2020. A survey comparing specialized hardware and evolution in TPUs for neural networks. In *IEEE 23rd International Multitopic Conference (INMIC'20)*. 1–6. DOI: https://doi.org/10.1109/INMIC50486.2020.9318136

[133] Junnan Shan, Mario R. Casu, Jordi Cortadella, Luciano Lavagno, and Mihai T. Lazarescu. 2019. Exact and heuristic allocation of multi-kernel applications to multi-FPGA platforms. In *56th Annual Design Automation Conference 2019*. 1–6.

[134] Hardik Sharma, Jongse Park, Emmanuel Amaro, Bradley Thwaites, Praneetha Kotha, Anmol Gupta, Joon Kyung Kim, Asit Mishra, and Hadi Esmaeilzadeh. 2016. DnnWeaver: From high-level deep network models to FPGA acceleration. In *Workshop on Cognitive Architectures*.

[135] A. Shawahna, S. M. Sait, and A. El-Maleh. 2019. FPGA-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access* 7 (2019), 7823–7859. DOI: https://doi.org/10.1109/ACCESS.2018.2890150

[136] Y. Shen, M. Ferdman, and P. Milder. 2017. Escher: A CNN accelerator with flexible buffering to minimize off-chip transfer. In *IEEE 25th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'17)*. 93–100. DOI: https://doi.org/10.1109/FCCM.2017.47

[137] Taylor Simons and Dah-Jye Lee. 2019. A review of binarized neural networks. *Electronics* 8, 6 (2019), 661.

[138] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556* (09 2014).

[139] Derya Soydaner. 2022. Attention mechanism in neural networks: Where it comes and where it goes. *Neural Comput. Applic.* (May 2022). DOI: https://doi.org/10.1007/s00521-022-07366-3

[140] Emanuele Del Sozzo, Davide Conficconi, Alberto Zeni, Mirko Salaris, Donatella Sciuto, and Marco D. Santambrogio. 2022. Pushing the level of abstraction of digital system design: A survey on how to program FPGAs. *Comput. Surv.* 55, 5 (2022), 1–48.

[141] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. 2021. Bottleneck transformers for visual recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16519–16529.

[142] Gilbert W. Stewart. 1993. On the early history of the singular value decomposition. *SIAM Rev.* 35, 4 (1993), 551–566.
[143] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16)*. Association for Computing Machinery, New York, NY, 16–25. DOI: https://doi.org/10.1145/2847263.2847276
[144] V. Sze, Y. Chen, T. Yang, and J. S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329. DOI: https://doi.org/10.1109/JPROC.2017.2761740
[145] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1–9. DOI: https://doi.org/10.1109/CVPR.2015.7298594
[146] Shinya Takamaeda-Yamazaki. 2015. Pyverilog: A Python-based hardware design processing toolkit for Verilog HDL. In *11th International Symposium on Applied Reconfigurable Computing*. Springer, 451–460.
[147] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *36th International Conference on Machine Learning (Proceedings of Machine Learning Research,* Vol. 97*)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6105–6114. Retrieved from https://proceedings.mlr.press/v97/tan19a.html
[148] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. 2020. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558* (2020).
[149] Yaman Umuroglu, Yash Akhauri, Nicholas James Fraser, and Michaela Blott. 2020. LogicNets: Co-designed neural networks and circuits for extreme-throughput applications. In *30th International Conference on Field-Programmable Logic and Applications (FPL'20)*. IEEE, 291–297.
[150] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, 65–74.
[151] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems Conference*.
[152] Stylianos I. Venieris and Christos-Savvas Bouganis. 2016. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *IEEE 24th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'16)*. IEEE, 40–47.
[153] Stylianos I. Venieris and Christos-Savvas Bouganis. 2017. fpgaConvNet: Automated mapping of convolutional neural networks on FPGAs. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 291–292.
[154] Stylianos I. Venieris and Christos-Savvas Bouganis. 2017. Latency-driven design for FPGA-based convolutional neural networks. In *27th International Conference on Field Programmable Logic and Applications (FPL'17)*. IEEE, 1–8.
[155] Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions. *arXiv preprint arXiv:1803.05900* (2018).
[156] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. 2016. DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 36, 3 (2016), 513–517.
[157] Erwei Wang, James J. Davis, Peter Y. K. Cheung, and George A. Constantinides. 2019. LUTNet: Rethinking inference in FPGA soft logic. In *IEEE 27th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM'19)*. IEEE, 26–34.
[158] Erwei Wang, James J. Davis, Peter Y. K. Cheung, and George A. Constantinides. 2020. LUTNet: Learning FPGA configurations for highly efficient neural network inference. *IEEE Trans. Comput.* 69, 12 (2020), 1795–1808.
[159] Jie Wang, Licheng Guo, and Jason Cong. 2021. AutoSA: A polyhedral compiler for high-performance systolic arrays on FPGA. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. 93–104.
[160] 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 *ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. 11–20.
[161] Tao Wang, Changhua Lu, Mei Yang, Feng Hong, and Chun Liu. 2020. A hybrid method for heartbeat classification via convolutional neural networks, multilayer perceptrons and focal loss. *PeerJ Comput. Sci.* 6 (2020), e324.
[162] Teng Wang, Chao Wang, Xuehai Zhou, and Huaping Chen. 2019. An overview of FPGA based deep learning accelerators: Challenges and opportunities. In *IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS'19)*. 1674–1681. DOI: https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00229
[163] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. 2016. DeepBurning: Automatic generation of FPGA-based learning accelerators for the neural network family. In *53rd Annual Design Automation Conference*. 1–6.
[164] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *54th Annual Design Automation Conference*. 1–6.

[165] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer. 2018. Shift: A zero FLOP, zero parameter alternative to spatial convolutions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9127–9135. DOI: https://doi.org/10.1109/CVPR.2018.00951

[166] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. 2020. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602* (2020).

[167] Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. 2022. TinyViT: Fast pretraining distillation for small vision transformers. In *17th European Conference ON Computer Vision (ECCV'22)*. Springer, 68–85.

[168] Ran Wu, Xinmin Guo, Jian Du, and Junbao Li. 2021. Accelerating neural network inference on FPGA-based platforms—A survey. *Electronics* 10, 9 (2021). DOI: https://doi.org/10.3390/electronics10091025

[169] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan, and Yu-Wing Tai. 2017. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In *54th Annual Design Automation Conference 2017*. 1–6.

[170] Xilinx. 2021. UltraScale Architecture DSP Slice (UG579). https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp

[171] Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, John Wawrzynek, et al. 2019. Synetgy: Algorithm-hardware co-design for ConvNet accelerators on embedded FPGAs. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. 23–32.

[172] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *Adv. Neural Inf. Process. Syst.* 35 (2022), 27168–27183.

[173] Maria Yatsenko. 2020. FPGAs for Artificial Intelligence: Possibilities, Pros, and Cons. Retrieved from https://www.apriorit.com/dev-blog/586-fpgas-for-ai

[174] Amir Yazdanbakhsh, Jongse Park, Hardik Sharma, Pejman Lotfi-Kamran, and Hadi Esmaeilzadeh. 2015. Neural acceleration for GPU throughput processors. In *48th International Symposium on Microarchitecture*. 482–493.

[175] H. Yonekawa and H. Nakahara. 2017. On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'17)*. 98–105. DOI: https://doi.org/10.1109/IPDPSW.2017.95

[176] Lu Yuan, Dongdong Chen, Yi-Ling Chen, Noel Codella, Xiyang Dai, Jianfeng Gao, Houdong Hu, Xuedong Huang, Boxin Li, Chunyuan L, Ce Liu, Mengchen Liu, Zicheng Liu, Yumao Lu, Yu Shi, Lijuan Wang, Jianfeng Wang, Bin Xiao, Zhen Xiao, Jianwei Yang, Michael Zeng, Luowei Zhou, and Pengchuan Zhang. 2021. Florence: A new foundation model for computer vision. *CoRR* abs/2111.11432 (2021). https://arxiv.org/abs/2111.11432

[177] Kun Zeng, Jun Yu, Ruxin Wang, Cuihua Li, and Dacheng Tao. 2015. Coupled deep autoencoder for single image super-resolution. *IEEE Trans. Cybern.* 47, 1 (2015), 27–37.

[178] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'15)*. Association for Computing Machinery, New York, NY, 161–170. DOI: https://doi.org/10.1145/2684746.2689060

[179] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2018. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Trans. Comput.-aid. Des. Integ. Circ. Syst.* 38, 11 (2018), 2072–2085.

[180] Jialiang Zhang and Jing Li. 2017. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network. In *ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. 25–34.

[181] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition*. 6848–6856.

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

[183] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 8697–8710.

Received 8 November 2022; revised 7 April 2023; accepted 25 July 2023