FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization

Mengshu Sun[1]*, Zhengang Li[1]*, Alec Lu[2]*, Yanyu Li[1], Sung-En Chang[1], Xiaolong Ma[1]
Xue Lin[1], Zhenman Fang[2]
[1]Department of Electrical & Computer Engineering, Northeastern University, Boston, MA, United States
[2]School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada
[1]{sun.meng, li.zhen, li.yanyu, chang.sun, ma.xiaol, xue.lin}@notheastern.edu
[2]{alec_lu, zhenman}@sfu.ca

ABSTRACT

With the trend to deploy Deep Neural Network (DNN) inference models on edge devices with limited resources, quantization techniques have been widely used to reduce on-chip storage and improve computation throughput. However, existing DNN quantization work deploying quantization below 8-bit may be either suffering from evident accuracy loss or facing a big gap between the theoretical improvement of computation throughput and the practical inference speedup.

In this work, we propose a general framework, called FILM-QNN, to quantize and accelerate multiple DNN models across different embedded FPGA devices. First, we propose the novel intra-layer, mixed-precision quantization algorithm that assigns different precisions onto the filters of each layer. The candidate precision levels and assignment granularity are determined from our empirical study with the capability of preserving accuracy and improving hardware parallelism. Second, we apply multiple optimization techniques for the FPGA accelerator architecture in support of quantized computations, including DSP packing, weight reordering, and data packing, to enhance the overall throughput with the available resources. Moreover, a comprehensive resource model is developed to balance the allocation of FPGA computation resources (LUTs and DSPs) as well as data transfer and on-chip storage resources (BRAMs) to accelerate the computations in mixed precisions within each layer. Finally, to improve the portability of FILM-QNN, we implement it using Vivado High-Level Synthesis (HLS) on Xilinx PYNQ-Z2 and ZCU102 FPGA boards. Our experimental results of ResNet-18, ResNet-50, and MobileNet-V2 demonstrate that the implementations with intra-layer, mixed-precision (95% of 4-bit weights and 5% of 8-bit weights, and all 5-bit activations) can achieve comparable accuracy (70.47%, 77.25%, and 65.67% for the three models) as the 8-bit (and 32-bit) versions and comparable throughput (214.8 FPS, 109.1 FPS, and 537.9 FPS on ZCU102) as the 4-bit designs.

* The first three authors contribute equally to this paper.

FPGA '22, February 27–March 1, 2022, Virtual Event, USA.

ACM ISBN 978-1-4503-9149-8/22/02…$15.00
https://doi.org/10.1145/3490422.3502364

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Hardware** → **Hardware accelerators**.

KEYWORDS

Deep learning, model compression, mixed-precision quantization, FPGA, hardware acceleration

ACM Reference Format: Mengshu Sun, Zhengang Li, Alec Lu, Yanyu Li, Sung-En Chang, Xiaolong Ma, Xue Lin, Zhenman Fang. 2022. FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '22), February 27–March 1, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3490422.3502364

1 INTRODUCTION

As DNNs have achieved a great success in various domains, they are increasingly deployed for inference on edge devices such as embedded FPGAs and ASICs. The growing amount of parameters and computations with complicated DNN architectures makes model compression an essential procedure for DNN acceleration on these edge devices, where the computation and storage resources are scarce. As one type of the key model compression techniques, model quantization has been broadly investigated at both algorithm-level [2–4, 6–10, 12, 17, 18, 22–26, 29, 34, 36, 39, 40, 44, 45, 49, 51, 54–56] and hardware-level [5, 14–16, 20, 28, 30–32, 35, 42, 43, 48, 52] to improve the inference throughput with the available resources.

Unfortunately, few studies have concentrated on both maintaining the model accuracy and enhancing the hardware computation throughput simultaneously. On one hand, many prior studies [6, 7, 18, 24, 25, 36, 56] achieve high throughput with low-precision (below 8-bit) quantization like binary, ternary, and 4-bit fixed-point quantization. However, the loss in their model accuracy becomes non-negligible. On the other hand, several studies [7, 20, 24, 36, 56] attempt to mitigate the accuracy loss by adopting inter-layer, mixed-precision quantization, i.e., assigning different precisions for different layers. This nonetheless leads to low hardware efficiency, since different layers of the inference now utilize different types of resources and they execute sequentially layer by layer. For example, on an FPGA platform, the first and last DNN layers with higher precision would mainly consume DSPs whereas middle DNN layers with lower precision would rely on look-up tables (LUTs). As a result, during the execution of one DNN layer, either DSPs or LUTs remain almost idle.

To achieve both high inference throughput and high model accuracy, we design and implement FILM-QNN, a general and resource-efficient **F**PGA acceleration framework for **I**ntra-**L**ayer, **M**ixed-Precision **Q**uantized D**NN**s. This is based on the observation that intra-layer, mixed-precision quantization, i.e., mixing higher precision and lower precision within each layer, has the potential to not only preserve the model accuracy but also better utilize all types of FPGA resources for concurrent computations, since computations with different precisions within a layer run simultaneously.

In FILM-QNN, we first propose a novel intra-layer, mixed-precision quantization algorithm that combines a majority of low precision (e.g., 95% of 4-bit) weights with a minority of high precision (e.g., 5% of 8-bit) weights within each layer, together with moderate precision (e.g., 5-bit) activations. To provide enough hardware parallelism, during the DNN training, we assign different precisions to the weights at the filter granularity, where weights potentially leading to high quantization errors are assigned with high precision. Second, for the FPGA accelerator design, we pipeline the accelerator with parallelization along the input channel and output filter dimensions, and employ multiple optimization techniques for quantized computations to improve the computation throughput. These optimizations include 1) DSP packing that efficiently packs multiple low-precision operations to share a single DSP, 2) weight reordering that arranges weights with the same precision together in each tile to eliminate the indexing overhead, and 3) data packing that packs multiple low-precision weights (or activations) to widen their data width, and so to save the on-chip storage and improve the data access parallelism. Finally, we build a comprehensive model to explore the FPGA resource allocation—including computation resources such as DSPs and LUTs, and on-chip memory such as Block RAMs (BRAMs)—to accelerate the computations with different precisions concurrently for an optimized overall throughput.

To demonstrate the portability of FILM-QNN, we have developed it with Xilinx Vivado High-Level Synthesis (HLS), and evaluated ResNet-18, ResNet-50, and MobileNet-V2 models on two embedded FPGA boards, Xilinx PYNQ-Z2 and ZCU102. Our optimized implementations—with 95% 4-bit and 5% 8-bit intra-layer, mixed-precision quantization—achieve comparable accuracy (70.47%, 77.25%, and 65.67% for the three models, respectively) to the pure 8-bit precision and original 32-bit full-precision designs, and comparable throughput (27.8 FPS (frames per second), 13.3 FPS, and 132.3 FPS on PYNQ-Z2, and 214.8 FPS, 109.1 FPS, and 537.9 FPS on ZCU102) to the pure 4-bit precision designs.

2 RELATED WORK

2.1 DNN Model Quantization

Model quantization is a crucial technique for DNN inference acceleration on edge devices such as embedded FPGAs and ASICs. Indeed, industry has already widely adopted the 8-bit quantization that can achieve almost the same inference accuracy of 32-bit floating-point based DNN models. Here we mainly discuss quantization techniques that use lower than 8-bit precision.

2.1.1 Uniform Extreme Low-Precision Quantization. Uniform extreme low-precision quantization mainly includes binary and ternary quantization, where each network value only takes one or two bits. Binary quantization maps values to $\{-1, +1\}$ represented using a single bit. Representative works include Binaryconnect [6], Binarized Neural Network (BNN) [7], XNOR-net [36], and ABC-Net [25]. On the other hand, ternary quantization in TWN [24], TTQ [56], and [18] uses zero as one more quantization level than the binary quantization, using two bits to represent values $\{-1, 0, +1\}$. However, the drawback of these uniform low-precision quantization methods is that they suffer from big accuracy loss at $> 5\%$ and 2%–3% for the binary quantization and ternary quantization.

2.1.2 Inter-Layer, Mixed-Precision Quantization. In previous work with low-precision quantization, a common practice to maintain the model accuracy is leaving the first and last layer unquantized, or quantizing them with no fewer than 8 bits [6, 17, 18, 37]. Later work [8, 9, 39, 40, 44, 45, 49] explored inter-layer, mixed-precision quantization methods to improve the quantization accuracy, which assign different precisions to different network layers. Different methods/algorithms have been utilized to explore the vast search space generated by layer-wise precision assignment of weights and activations, such as HAQ [44], DNAS [45], Mixed Precision DNNs [40]. Furthermore, HAWQ [9], HAWQ-V2 [8], PyHessian [49] and Q-BERT [39] solve Hessian matrices to find the optimized bit-width assignment of each layer, adding more bits to layers sensitive to the quantization errors. One major issue with inter-layer, mixed-precision quantization is that it could lead to low hardware efficiency, as explained in Section 1 and evaluated in Section 4.

2.1.3 Intra-Layer, Mixed-Precision Quantization. To further elaborate on the concept of mixed-precision quantization, several studies [26, 34] explore intra-layer, mixed-precision quantization to enable different precisions or schemes within each layer. For example, RVQuant [34] proposes value-aware quantization, which only applies low precision quantization to small data in weights and activations and achieves better model accuracy. However, it cannot reach remarkable inference speedups compared with the uniform low-precision quantization. AutoQ [26] utilizes reinforcement learning to determine the quantization precision within kernel level, which is computationally expensive.

In this paper, we focus on the intra-layer, mixed-precision quantization and further explore its potential to improve the hardware efficiency. Unlike prior studies, our goal is to both preserve the model accuracy and improve the inference throughput.

2.2 FPGA Acceleration of Quantized DNNs

In ASIC designs, dynamic quantization with bit fusion [38] and stripes bit-serial units [21] improve the bit-level flexibility by matching different bit-widths across DNN layers. Likewise, model quantization has been widely adopted into DNN implementations on FPGAs [14]. Most of existing studies focus on binary quantization, executing computations of binarized weights and activations with XNOR gates [16, 30, 31, 41, 53]. The work in [20] utilizes a two-stage arithmetic unit realized by LUTs for 2-bit quantization, while preserving 8-bit data for the first convolutional layer. Power-of-two (PoT) quantized model deployment is investigated in [28] with LUTs for shift-accumulation operations.

More approaches have been explored for fixed-point quantization, such as mixed-precision quantization in [43] for inter-layer

mixed weight bit-widths (1, 2, 4, or 8 bits), software-hardware co-design for 1-bit weights and 4-bit activations in [48], greedy solution in [15] to determine the radix position of each layer, and automatic generation of RTL DNN components in [52] for higher precision (8 and 16 bits). The OpenCL-based deep learning accelerator in [5] can accommodate DNNs with a wide range of bit-widths.

Unlike these prior studies, our work focuses on the FPGA acceleration of DNNs using the intra-layer, mixed-precision quantization, with the goal to accelerate the inference throughput and preserve the model accuracy. This is essential considering that it is often not possible to map the entire model (i.e., all layers) of large DNNs for on-chip processing simultaneously. The closest work to ours is the one in [2], which uses mixed quantization schemes (fixed-point and power-of-two quantizations) whereas we use mixed-precision (high and low bit-width) quantization within each layer. Moreover, we apply multiple optimization techniques for low-precision computations on FPGAs and develop a comprehensive model to balance the FPGA resource allocation in this work. A quantitative comparison to [2] will be presented in Section 4.4.

In close association with model quantization, a balanced resource utilization is also indispensable for FPGA implementations to achieve high computation throughput. For example, a design space exploration framework presented in [35] investigates architectural choices with various resource allocation of DSPs, on-chip memory and off-chip bandwidth to maximize the performance for DNN designs with 16-bit fixed-point precision. However, it lacks analysis for the LUT usage; as such, the computation for quantization bit-width below 11 is always implemented in LUTs. For quantization models with low precision (i.e., less than 8-bit), this approach would result in under utilization of DSPs on FPGAs. In our work, we model the computation using both LUTs and DSPs (as well as on-chip memory and off-chip bandwidth), with the goal to balance the resource allocation between different precisions and utilize all tyes of resources simultaneously. Moreover, we also apply the DSP packing technique as described in [32, 42, 46, 47] to fully utilize DSPs for efficient computation of low-precision operands.

3 PROPOSED FILM-QNN FRAMEWORK

3.1 Accurate and Hardware-Friendly Intra-Layer, Mixed-Precision Quantization

Mixed-precision quantization has been extensively explored for preserving the accuracy of quantized models, while reducing the computation amount to accelerate inference. However, the existing inter-layer approaches [8, 9, 39, 40, 44, 45, 49] assign different precisions onto the layers of a DNN, and therefore are not compatible with the layer-by-layer inference execution on hardware platforms. On the other hand, the existing intra-layer approaches [26, 34] use a much finer granularity for precision assignment within layers, and also incur execution overhead for DNN inference. *Therefore, there exists a big gap between the (theoretical) reduction of computation amount and the (practical) inference speedup.*

Our intra-layer, mixed-precision quantization is motivated from the following observations: 1) It is well known that the 8-bit quantized inference models can easily preserve the accuracy as the full-precision models [13]. 2) The accuracy drop caused by a low bit-width quantization can be mitigated when mixing with a high bit-width quantization in a certain ratio. Here, 8-bit quantization can serve as the high bit-width one, due to its superior accuracy performance. This is why even in uniform low bit-width quantization approaches [3, 4, 10, 12, 17, 22, 55], the first and last layers are quantized into no fewer than 8-bit or even left without quantization.

Spurred by the above observations, we propose our novel intra-layer, mixed-precision quantization. Specifically, we propose to combine the *low* and *high* bit-width quantization *within layers* and *in a proper granularity* to preserve accuracy as well as to reduce execution overhead during the layer-by-layer inference computations. Note that it is often not possible to map the entire model of large DNNs onto an FPGA and execute all layers simultaneously. We set the granularity of intra-layer quantization assignment down to the filter level to ensure hardware parallelism. Additionally, we use a *moderate* bit-width quantization for *all* the activations in the whole DNN model as a proper choice for preserving accuracy.

In summary, according to our empirical study (with results presented in Section 4.2), we adopt a total of three precision levels in our intra-layer, mixed-precision quantization approach: (I) a low bit-width (4-bit) for majority of the filters in a layer, (II) a high bit-width (8-bit) for $R = 5\%$ of the filters in a layer, and (III) a moderate bit-width (5-bit) for all the activations throughout the model, to preserve accuracy and to reduce execution overhead for inference acceleration. In Section 4.2, we will explore the impact of different R values (i.e., the percentage of high bit-width filters) on the accuracy.

Algorithm 1 shows our novel intra-layer, mixed-precision quantization approach for quantizing the model to satisfy the above-mentioned three bit-width constraints (I), (II), and (III). Specifically, to determine the precision assignment for the filters (rows) in the weight tensor (matrix) of each convolutional (fully-connected) layer, we propose to calculate the quantization error $\mathbb{E}$. In Algorithm 1, the proposed precision assignment process is integrated with the Straight Through Estimator (STE) [1, 2, 55], a quantization-aware training method. Details of our approach are provided as follows.

Consider an L-layer DNN with both convolutional and fully-connected layers. The weight tensor (matrix) of the l-th layer is denoted as $\mathbf{W}_l$. The activation of the l-th layer is denoted by $\mathbf{A}_l$. We use $\mathbf{W}_l^k$ to denote the k-th filter (row) in the weight tensor (matrix) $\mathbf{W}_l$. With m-bit fixed-point quantization, the quantized weight and activation values are derived from

$$\mathbb{V}(m) = \pm\alpha \times \{0, \frac{1}{2^{m-1}-1}, \frac{2}{2^{m-1}-1}, \ldots, 1\}, \tag{1}$$

where α is the scaling factor. For weight quatization, α is calculated as the maximum absolute value of all weights in each layer, while for activation quantization, α is a fixed value related to the dataset. Then the quantizer function from a original floating-point value x to an m-bit fixed-point value $\hat{x}$ is expressed as

$$\hat{x} = \mathbb{Q}_{mb}(x) = \alpha \cdot h^{-1}\left(\frac{1}{2^m-1} \cdot \text{round}\big((2^m-1) \cdot h\big(\lceil x, \alpha \rfloor\big)\big)\right), \tag{2}$$

where $h(\cdot)$ shifts a value within $[-1, +1]$ into the range of $[0, 1]$ (e.g., $h(x) = x/2 + 0.5$), and $\lceil x, \alpha \rfloor$ clips x by

$$\lceil x, \alpha \rfloor = \begin{cases} -1, & x < -\alpha \\ x/\alpha, & -\alpha \le x \le \alpha \\ 1, & x > \alpha \end{cases}. \tag{3}$$

Algorithm 1: The proposed novel intra-layer, mixed-precision quantization

input : A DNN model $\mathcal{M}$ in 32-bit floating-point, with the l-th layer weight tensor $\mathbf{W}_l$ $(1 \le l \le L)$;
The target ratio R for High bit-width weights;

output : The quantized model $\hat{\mathcal{M}}$, where each layer has $1-R$ of the filters in Low bit-width quantization $\mathbb{Q}_{\text{Lb}}$ and R of the filters in High bit-width quantization $\mathbb{Q}_{\text{Hb}}$, and all the activations are in Moderate bit-width quantization $\mathbb{Q}_{\text{Mb}}$;

1 **foreach** *epoch* **do**
2 **foreach** *batch* **do**
  // forward propagation
3 **foreach** *layer* l **do**
  // calculate quantization error for each filter in Low bit-width quantization at first batch of every epoch
4 **foreach** *filter* $\mathbf{W}_l^k$ *in* $\mathbf{W}_l$ **do**
5 $\text{quan\_error}_k \leftarrow \mathbb{E}(\mathbf{W}_l^k, \hat{\mathbf{W}}_l^k)$ in $\mathbb{Q}_{\text{Lb}}$;
  // assign quantization precision for each filter in the layer
6 **foreach** *filter* $\mathbf{W}_l^k$ *in* $\mathbf{W}_l$ **do**
7 $\hat{\mathbf{W}}_l^k \leftarrow \mathbb{Q}_{\text{Hb}}(\mathbf{W}_l^k)$ if quan_error_k belongs to the top R (5%) group;
8 $\hat{\mathbf{W}}_l^k \leftarrow \mathbb{Q}_{\text{Lb}}(\mathbf{W}_l^k)$ otherwise;
9 $\mathbf{A}_l \leftarrow \hat{\mathbf{W}}_l \cdot \hat{\mathbf{A}}_{l-1}$;
  // activations are quantized in Moderate bit-width
10 $\hat{\mathbf{A}}_l \leftarrow \mathbb{Q}_{\text{Mb}}(\mathbf{A}_l)$;
  // backward propagation
11 **foreach** *layer* l *(reverse order)* **do**
12 $\frac{\partial Loss}{\partial \mathbf{W}_l} \leftarrow \frac{\partial Loss}{\partial \hat{\mathbf{W}}_l}$;
13 $\frac{\partial Loss}{\partial input_l} \leftarrow \frac{\partial Loss}{\partial \hat{\mathbf{A}}_{l-1}}$;
14 Return $\hat{\mathcal{M}} \leftarrow \mathcal{M}\{\hat{\mathbf{W}}\}$.

We propose the quantization error of a filter $\mathbf{W}_l^k$ as:

$$\mathbb{E}(\mathbf{W}_l^k, \hat{\mathbf{W}}_l^k) = \|\mathbf{W}_l^k \cdot \hat{\mathbf{A}}_{l-1} - \hat{\mathbf{W}}_l^k \cdot \hat{\mathbf{A}}_{l-1}\|_2 \tag{4}$$

where $\|\cdot\|_2$ means the L_2 norm. The quantization error measures the difference between output feature maps generated by the full-precision filter and those by the quantized filter. To assign precision for each filter in a layer, we calculate their quantization errors as if they are quantized in Low bit-width (line# $4-5$). If the quantization error of a filter belongs to the top R (e.g., 5%) group, this filter should be quantized in High bit-width; otherwise, it should be quantized in Low bit-width (line# $6-8$). As seen in Algorithm 1, our precision assignment process (line# $4-8$) is nested within the STE method, which uses rounded weights/activations in the forward propagation, and solves the unavailable gradient issue during the backward propagation by setting the derivative of rounding function as **1**.

3.2 FPGA Accelerator Design and Optimization

Our base FPGA accelerator design is similar to [50] with the support of various types of DNN layers including convolutional layer, fully-connected layer, batch normalization layer, activation (ReLU) layer, and pooling layer, as well as skip connection (namely identity shortcut connection). For FILM-QNN, we introduce the extra optimizations for quantized computations in this subsection, including DSP packing, weight reordering, and data packing.

Algorithm 2: Tiled convolution accelerator

input : Input tile I with size of $T_n \times T_r^{in} \times T_c^{in}$
Weight tile W with size of $T_m \times T_n \times K_r \times K_c$

output : Output tile O with size of $T_m \times T_r \times T_c$

// Notations of T_m (T_n), T_r (T_c), T_r^{in} (T_c^{in}), K_r (K_c), and S_r (S_c) are defined in Table 1.
// Input tile I and output tile O are obtained from activations A_{l-1} and A_l, respectively.

Convolution(I, W, O) {
  for $(k_r = 1 : K_r, k_c = 1 : K_c)$ // kernel
    L1: **for** $(t_r = 1 : T_r, t_c = 1 : T_c)$ { // feature map
#PIPELINE II=1
      L2: **for** $(t_m = 1 : T_m)$ { // filter
#UNROLL
        L3: **for** $(t_n = 1 : T_n)$ { // input channel
#UNROLL
          $O[t_m][t_r][t_c]$ += $W[t_m][t_n][k_r][k_c] \times I[t_n][t_r \times S_r + k_r][t_c \times S_c + k_c]$;
}}}}

Base FPGA Accelerator. Without loss of generality, we use the most computation-intensive convolutional layer to demonstrate our FPGA design details. Given the limited FPGA resources, we apply the commonly used tiling technique in the accelerator implementation, where the accelerator only computes for a single tile of input feature maps, weight kernels, and output feature maps from a layer at a time. Algorithm 2 provides the pseudo code of the tiled convolution accelerator design. Details are explained as follows.

We use T_m (T_n), T_r (T_c), T_r^{in} (T_c^{in}), K_r (K_c) and S_r (S_c) to denote the dimensionalities of the tiled convolution procedure with explanations provided in Table 1. The tiles of input feature maps and weights are loaded from off-chip memory to on-chip Block RAM (BRAM) buffers in a burst mode, and then processed by the tiled convolution procedure (Algorithm 2), which computes the tile of corresponding output feature maps. Lastly, the output tile data on BRAM is written back to the off-chip memory in a burst mode.

Inside the tiled convolution procedure (Algorithm 2), Loop L1 is pipelined with initial interval (II) of 1 to achieve computation parallelism of $T_m \times T_n$ in Loops L2 (filter, or output channel dimension) and L3 (input channel dimension). The buffered tiles (in BRAMs) are partitioned in the corresponding dimensions to support the parallelism: the input tile I is completely partitioned along the first dimension (input channel dimension), the output tile O is completely partitioned along the first dimension (filter dimension), and the weight tile I is completely partitioned along the first and second dimensions (both filter and input channel dimensions). In addition, double buffering is applied to pipelining the following stages: loading input and weight tiles, computing the convolution, and storing the output tile.

3.2.1 DSP Packing for Multiple Quantized Multiplications. To fully exploit the potential of DSP resources on FPGAs, we pack multiple

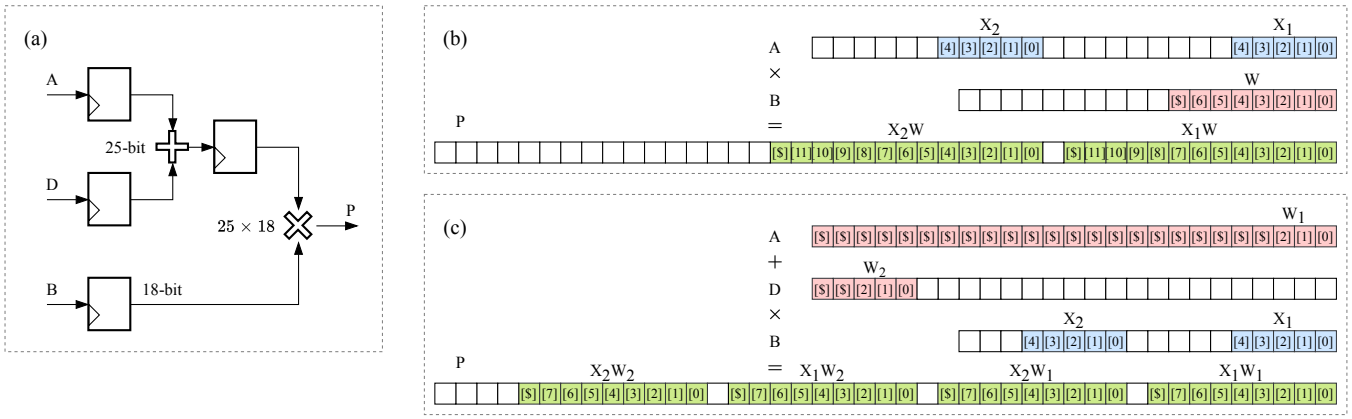

Figure 1: Multiple quantized multiplications supported on a single DSP48E1. (a) The 25×18-bit multiplication supported on DSP48E1; (b) Two 8×5-bit multiplications packed on DSP48E1; and (c) Four 4×5-bit multiplications packed on DSP48E1.

Table 1: Notations used in FILM-QNN

Notation	Description
T_m (T_n)	Tiling size in output (input) channel dimension
T_r (T_c)	Tiling size in output feature height (width) dimension
T_r^{in} (T_c^{in})	Tiling size in input feature height (width) dimension
K_r (K_c)	Kernel size in height (width) dimension
S_r (S_c)	Stride in height (width) dimension
$C_{lut}^{4\times5,dsp}$ ($C_{lut}^{8\times5,dsp}$)	Number of LUT used by a 4×5-bit (or 8×5-bit) multiplication executed on DSPs
$C_{dsp}^{4\times5}$ ($C_{lut}^{8\times5}$)	Number of DSP (LUT) used by multiplications executed on DSPs (LUTs)
G	Number of low-bit data packed together
$N_{dsp}^{4\times5}$ ($N_{lut}^{8\times5}$)	Number 4×5-bit (8×5-bit) multiplications executed on DSPs (LUTs)
S_{dsp} (S_{lut}, S_{bram})	Available number of DSPs (LUTs, BRAMs)
γ_{dsp} (γ_{lut})	DSP (LUT) utilization threshold
B_{in} (B_{out}, B_{wgt})	Number of BRAMs used by input (output, weight) tile
p_{in} (p_{out}, p_{wgt})	Number of AXI ports used for data transfer of input (output, weight) tile
C_{cmpt} (C_{wgt}, C_{in})	Number of computation (weight transfer, input transfer) cycles for one group of tiles
S_{port}	Size of one AXI port on FPGA

low-precision multiplications within each DSP following [46, 47]. Each DSP (DSP48E1) on the PYNQ-Z2 board can handle one 25×18-bit multiplication (and one 25-bit addition), whereas each DSP (DSP48E2) on the ZCU102 board could handle one 27×18-bit multiplication. We take the DSP48E1 (25×18-bit) as an example to describe our DSP packing technique. In our FILM-QNN, we adopt two precisions for weights, i.e., 4-bit and 8-bit, and one precision for activations, i.e., 5-bit. Therefore, we need to support two types of multiplications i.e., 4×5-bit and 8×5-bit. As shown in Fig. 1(a), a DSP48E1 can support the computation of $P = (A + D) \times B$, where both A and D are 25-bit operands, B is an 18-bit operand, and the P is the 43-bit output. As shown in Fig. 1(b) and Fig. 1(c), a DSP48E1 can support *two* 8×5-bit multiplications or *four* 4×5-bit multiplications.

To pack two 8×5-bit multiplications in a DSP48E1, shown in Fig. 1 (b), two 5-bit activations X_1 and X_2 are presented in the 25-bit operand A, occupying the bits $[4:0]$ and $[18:14]$, and one 8-bit signed weight W is presented in the 18-bit operand B occupying its bits $[7:0]$, where [\$] denotes the sign of the weight. The two 13-bit multiplication results can be extracted from the bits $[12:0]$ and $[26:14]$, respectively, in the 43-bit product P of the DSP. The bit $[13]$ in the product P is reserved for overflow. To reduce the LUT consumption for additions, only lower bits of these operands and output are occupied.

To pack four 4×5 multiplications in a DSP48E1, shown in Fig. 1 (c), two 4-bit signed weights W_1 and W_2 are presented in the 25-bit operands, occupying the bits $[4:0]$ of operand A and $[23:20]$ of operand D to properly extend and carry the sign bits of the weights, while two 5-bit activations X_1 and X_2 are presented in the 18-bit operand B, occupying its bits $[4:0]$ and $[14:10]$. Four 9-bit products are generated in the bits $[8:0]$, $[18:10]$, $[28:20]$, and $[38:30]$ of the 43-bit product P of the DSP. Similarly, a bit for overflow is reserved between the generated four products.

While we illustrate the DSP packing on Xilinx FPGAs, similar techniques can also be performed on Intel FPGAs that have 18×18-bit and 27×27-bit fixed-point DSPs [19].

3.2.2 Weight Reordering to Reduce Indexing Overhead. Our intra-layer, mixed-precision quantization algorithm determines which (output) filters in a layer to assign 8-bit quantization. However, the $R = 5\%$ of High-bit (8-bit) filters may be distributed irregularly over the whole layer, as shown in the left part of Fig. 2. Since the 4-bit and 8-bit computations are processed separately (and concurrently) by different compute engines, this could result in the (random) indexing overhead to access these 8-bit and 4-bit filters in these compute engines. To reduce such indexing overhead, we propose the weight reordering technique as shown in Fig. 2. Suppose a layer has a total of M filters, which are divided into multiple weight tiles, each covering T_m filters. For each weight tile, we reorganize the filter ordering such that the first $R \cdot T_m$ filters are preserved for 8-bit quantized ones, and the rest filters in the tile are for 4-bit quantized ones. Correspondingly, the parameters in the subsequent batch normalization layer and the input channels in the next layer are rearranged with the same order as the filters in this layer.

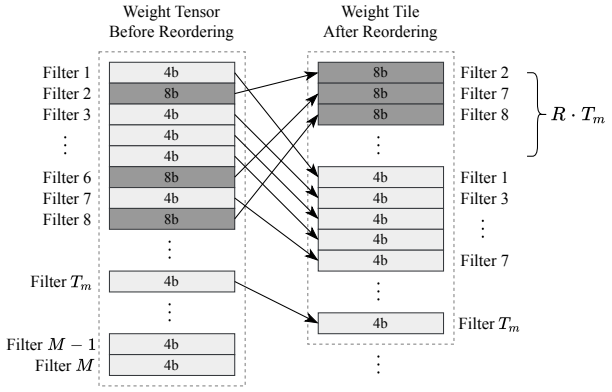

Figure 2: Weight filter reordering after quantization. Each filter (block in the figure) has three dimensions with total size of $T_n \times K_r \times K_c$.

This weight reordering technique is applied in compilation time before running the DNN inference on hardware, which eliminates the indexing overhead for layers that are sequentially connected. A special case is for the layers with skip connections, where the filters in the two skip-connected layers l_a and l_b might be reordered differently. This could be handled by storing the filter order of layer l_a relative to that of reordered layer l_b, and adjusting the filter order when loading the activation outputs of layer l_a for addition operations with those of layer l_b. Considering all filters in all layers, our weight reordering technique could reduce the indexing overhead from 67.97 kbit to 27.75 kbit (2.45×) for ResNet-18, from 323.0 kbit to 198.8 kbit (1.62×) for ResNet-50, and from 211.6 kbit to 9.28 kbit (22.8×) for MobileNet-V2.

3.2.3 Data Packing to Reduce Storage and Improve Parallelism. The low-precision data access in FILM-QNN (i.e., 4-bit and 8-bit for weights, and 5-bit for activations) could lead to the under-utilization of BRAM capacity and over-use of the BRAM banks (for parallel port access), since the bit-width of a BRAM bank (18k bits) is much wider. To save the on-chip BRAM usage and improve the data access parallelism on hardware, we apply the data packing optimization to pack multiple low-precision data items into one wide data item. The data packing size G is determined according to the available computation and memory resources on a specific FPGA board. As a result, we can save the BRAM usage and improve the on-chip bandwidth by G times when G is no larger than $BRAM\_bit\_width/low\_bit\_width$. In addition, this also improves the effective off-chip memory bandwidth by G times [27] since each off-chip memory access port is 64-bit wide on PYNQ-Z2 board and 128-bit wide on ZCU102 board. Note this data packing can be applied to Intel FPGAs as well.

For the input and output tiles, G channels of 5-bit activation values are packed as one. For the input tile I, the $T_n \times T_r^{in} \times T_c^{in}$ values in 5-bit representation are stored as $T_n/G \times T_r^{in} \times T_c^{in}$ data in $5 \cdot G$-bit representation. Similarly, the output tile O have $T_m/G \times T_r \times T_c$ data in $5 \cdot G$-bit. As for the weight tile W, the data packing is performed in the input channel dimension. Combining every two 4-bit weights into one 8-bit weight, the buffer stores $T_m^{wgt} \times T_n/G \times K_r \times K_c$ data in 8-bit, where $T_m^{wgt} = T_m/2 \cdot (1 + R)$. The required data in computations are extracted through selecting the corresponding range of packed data with shifting and AND operations.

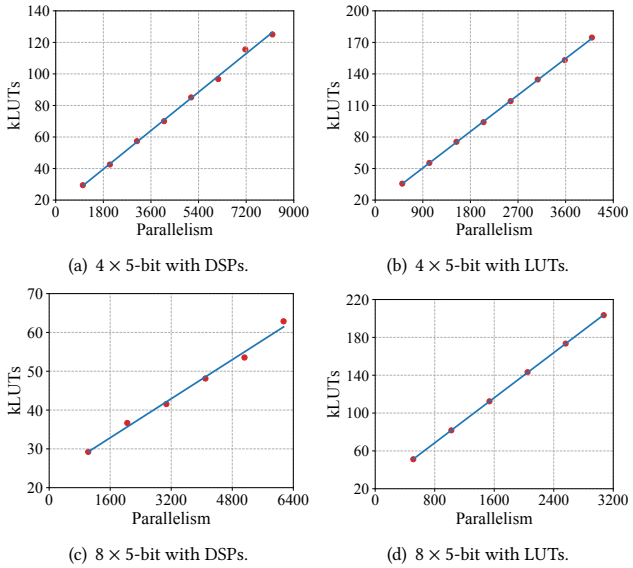

Figure 3: Number of utilized LUTs vs. parallelism on ZCU102 FPGA for (a) 4×5-bit multiplications on DSPs, (b) 4×5-bit multiplications on LUTs, (c) 8×5-bit multiplications on DSPs, and (d) 8×5-bit multiplications on LUTs.

3.2.4 Processing of Other Layers. The convolutional and fully-connected layers in DNNs could share the same computing kernel for matrix multiplication operations in the hardware implementation. Compared with matrix multiplication operations, the batch normalization, activation (ReLU), pooling, and skip connection (addition operations) in DNNs cost much fewer computation resources. These operations are executed before storing the results of the output buffers with negligible impact on the overall latency. For batch normalization layers, the parameters are kept with floating-point precision during model quantization. In the hardware implementation, the running mean and running variance parameters are fused into the weights and biases in batch normalization. The scaling factor α in each convolutional layer, mentioned in equation (1), is further fused into the weights and biases in the corresponding batch normalization layer, and these fused parameters are represented with sufficiently high precision (8-bit or 16-bit) to preserve the model accuracy. If there exists a skip connection, the results from this batch normalization layer and those from a previous ReLU layer would be added together. The activation values are quantized after each ReLU layer that performs comparison operations, followed by a pooling layer if exists. For this reason, representing the activation values with 5-bit numbers incurs negligible accuracy loss.

3.3 FPGA Resource Allocation Modeling

With our base FPGA accelerator design for quantized DNN models, we need to further improve the computation parallelism with the available computing resources (i.e., DSPs and LUTs), on-chip memory (i.e., BRAM), and memory bandwidth. However, we find it is difficult to estimate the LUT cost for given computation type (i.e., 4×5-bit or 8×5-bit multiplication) and even with specified computing resources (i.e., DSPs or LUTs). Therefore, we need to

first characterize the LUT consumption for the four cases in Fig. 3, i.e., (a) 4×5-bit multiplications executed on DSPs, (b) 4×5-bit multiplications executed on LUTs, (c) 8×5-bit multiplications executed on DSPs, and (d) 8×5-bit multiplications executed on LUTs. Based on this, we can use the slopes of the fitted lines as the LUT cost for a 4×5-bit multiplication executed on DSPs or LUTs, denoted by $C_{lut}^{4\times5,dsp}$ or $C_{lut}^{4\times5}$. Similarly, we also derive $C_{lut}^{8\times5,dsp}$, and $C_{lut}^{8\times5}$. It is worth noting that employing DSPs for quantized multiplications consumes LUTs as well, resulting from data packing and operations like accumulation. Next, we use a convolutional layer to show the details of our FPGA resource modeling and optimization, including the analysis on computing resources as well as off-chip data transfer and on-chip memory.

3.3.1 Computation Resource Analysis. From the computation resource utilization perspective, we would like to maximize the total number of concurrent 8×5-bit and 4×5-bit multiplications on both DSPs and LUTs as

$$\text{maximize} \left\{ N_{dsp}^{8\times5} + N_{lut}^{8\times5} + N_{dsp}^{4\times5} + N_{lut}^{4\times5} \right\} \tag{5}$$

where $N_{dsp}^{8\times5}$ denotes the number of 8×5-bit multiplications executed by DSPs, and descriptions for $N_{lut}^{8\times5}$, $N_{dsp}^{4\times5}$, and $N_{lut}^{4\times5}$ are in Table 1. We need to satisfy the following computation resource constraints

$$N_{dsp}^{8\times5} \cdot C_{dsp}^{8\times5} + N_{dsp}^{4\times5} \cdot C_{dsp}^{4\times5} \le S_{dsp} \cdot \gamma_{dsp} \tag{6}$$

$$\begin{aligned} &N_{lut}^{8\times5} \cdot C_{lut}^{8\times5} + N_{lut}^{4\times5} \cdot C_{lut}^{4\times5} + \\ &N_{dsp}^{8\times5} \cdot C_{lut}^{8\times5,dsp} + N_{dsp}^{4\times5} \cdot C_{lut}^{4\times5,dsp} \le S_{lut} \cdot \gamma_{lut} \end{aligned} \tag{7}$$

$$\frac{N_{dsp}^{8\times5} + N_{lut}^{8\times5}}{N_{dsp}^{8\times5} + N_{lut}^{8\times5} + N_{dsp}^{4\times5} + N_{lut}^{4\times5}} \ge R. \tag{8}$$

The above constraints (6) and (7) ensure the DSP and LUT utilization is under the allowable threshold, i.e., γ_{dsp} and γ_{lut} with the total resource amounts denoted by S_{dsp} and S_{lut}. And the constraint (8) makes sure we have no less than $R = 5\%$ for 8×5-bit multiplications. Depending on the characteristics of the target FPGA device, the final solution may converge to one of the four boundary conditions, which is determined by the output combination of two factors: 1) whether the 8×5-bit multiplications can be more efficiently processed on DSPs or LUTs on FPGAs; and 2) whether the available DSP (or LUT) resources are sufficient to handle those 8×5-bit multiplications. For each of these four boundary conditions, the situation is described when one of the four parameters ($N_{dsp}^{8\times5}$, $N_{dsp}^{4\times5}$, $N_{lut}^{8\times5}$, and $N_{lut}^{4\times5}$) is zero. The solution process for these four boundary conditions is similar. Due to space limitation, here we only present the solution for the case where the 8×5-bit multiplication is more efficiently processed on LUTs than on DSPs, and the available LUT resources are sufficient to handle those 8×5-bit multiplications, i.e.,

$$\begin{cases} \dfrac{C_{dsp}^{4\times5}}{C_{dsp}^{8\times5}} \le \dfrac{C_{lut}^{4\times5} - C_{lut}^{4\times5,dsp}}{C_{lut}^{8\times5} - C_{lut}^{8\times5,dsp}} \\ \dfrac{S_{dsp} \cdot \gamma_{dsp}}{q \cdot S_{lut} \cdot \gamma_{lut}} \le \dfrac{C_{dsp}^{4\times5}}{q \cdot C_{lut}^{4\times5,dsp} + C_{lut}^{8\times5}} \end{cases} \tag{9}$$

where $q = (1 - R)/R$. The solution is then expressed as

$$\begin{cases} N_{dsp}^{8\times5} = 0 \\ N_{dsp}^{4\times5} = \dfrac{S_{dsp}\gamma_{dsp}}{C_{dsp}^{4\times5}} \\ N_{lut}^{8\times5} = \dfrac{(C_{lut}^{4\times5} - C_{lut}^{4\times5,dsp})S_{dsp}\gamma_{dsp} + C_{dsp}^{4\times5}S_{lut}\gamma_{lut}}{qC_{lut}^{4\times5}C_{dsp}^{4\times5} + C_{dsp}^{4\times5}C_{lut}^{8\times5}} \\ N_{lut}^{4\times5} = \dfrac{qC_{dsp}^{4\times5}S_{lut}\gamma_{lut} - (qC_{lut}^{4\times5,dsp} + C_{lut}^{8\times5})S_{dsp}\gamma_{dsp}}{qC_{lut}^{4\times5}C_{dsp}^{4\times5} + C_{lut}^{8\times5}C_{dsp}^{4\times5}}. \end{cases} \tag{10}$$

3.3.2 Off-chip Data Transfer and On-chip Memory Analysis. During the inference execution of the FPGA accelerator, weight tiles need to be transferred through burst access from the off-chip DRAM memory to the on-chip BRAM buffers. The transferred tiles include input tile with size $T_n \times T_r^{in} \times T_c^{in}$, output tile with size $T_m \times T_r \times T_c$, and weight tile with size $T_m \times T_n \times K_r \times K_c$ as mentioned in Section 3.2. Here, we would like to maximize the hardware parallelism from the data access perspective, i.e.,

$$\underset{T_m, T_n, p_{in}, p_{wgt}}{\text{maximize}} \quad T_m \times T_n \tag{11}$$

where p_{wgt} and p_{in} are the numbers of ports used for loading weight and input buffers. This optimization problem subjects to the following constraints:

$$2 \cdot (B_{out} + B_{in} + B_{wgt}) \le S_{BRAM} \tag{12}$$

$$\max\{C_{in}, C_{wgt}\} \le C_{cmpt} \tag{13}$$

Constraint (12) ensures sufficient on-chip memory storage with double buffering, where B_{in}, B_{out} and B_{wgt} respectively denote the number of BRAMs used by input, output and weight tiles, and can be calculated by

$$\begin{aligned} B_{in} &= T_n/G \cdot \lceil T_r^{in} \cdot T_c^{in} \cdot 5 \cdot G/18\text{k} \rceil \\ B_{out} &= T_m/G \cdot \lceil T_r \cdot T_c \cdot 5 \cdot G/18\text{k} \rceil \\ B_{wgt} &= T_m/2 \cdot (1 + R) \cdot T_n/G \cdot \lceil K_r \cdot K_c \cdot 8 \cdot G/18\text{k} \rceil \end{aligned} \tag{14}$$

where the BRAM block size of 18Kb is used and the BRAM usage for each buffer rounds up to the nearest whole number ($\lceil \cdot \rceil$). Also note that the double buffering technique is used to overlap computations with off-chip memory accesses.

Constraint (13) is on off-chip data transfer, where C_{cmpt}, C_{wgt} and C_{in} are the number of computation cycles, weight transfer cycles, and input transfer cycles for one group of tiles, respectively. The output buffer transfer cycles are negligible and thus not included. C_{cmpt}, C_{wgt} and C_{in} are obtained as

$$\begin{aligned} C_{cmpt} &= K_r \cdot K_c \cdot T_r \cdot T_c \\ C_{wgt} &= \frac{T_m \cdot T_n \cdot K_r \cdot K_c \cdot (8 \cdot R + 4 \cdot (1 - R))}{p_{wgt} \cdot S_{port}} \\ C_{in} &= \frac{T_n \cdot T_r^{in} \cdot T_c^{in} \cdot 5}{p_{in} \cdot S_{port}} \end{aligned} \tag{15}$$

where S_{port} denotes the bit-width of one AXI port.

3.3.3 Summary. Ideally, we need to jointly solve both problems (5) and (11) for optimized FPGA resource allocation. In fact, we find that we only need to solve problem (5) and then check with the constraints of problem (11), because problem (11) generally provides a higher parallelism result for most of FPGAs. That is, the achievable parallelism degree in our quantized DNNs is generally bounded by the available computation resources on FPGAs, not the off-chip data transfer or on-chip memory resources.

4 EVALUATION

4.1 Experimental Setup

Our experiments include model quantization and hardware implementations for DNN models of different sizes, namely ResNet-18, ResNet-50 and MobileNet-V2. The baseline models with 32-bit floating-point (FP32) precision are downloaded from the TorchVision library [11] and we also retrain them with FP32 to get higher accuracy. Then we quantize them with intra-layer, mixed-precision for weights, where we explore different bits for the weights (4 and 8 bits) and activations (3 to 6 bits), and different ratios of 8-bit weight values (R = 0%, 3%, 5%, 10%, 20%, 100%). The 5% 8-bit filters are determined in the first 100 epochs, and the model parameters are finetuned in the remaining 50 epochs with the precision fixed.

The initial learning rate is 1.024 for ResNet-50, and 0.512 for ResNet-18 and MobileNet-V2. The batch size is set as 1024 for ResNet-50 and 512 for the other two models. The SGD optimizer is used with momentum as 0.875 and weight decay as 1/32768 to train the models. As for training tricks to improve accuracy, 8 epochs of warmup training are performed at the beginning of the quantization and label smoothing is applied with a factor of 0.1. These are the image classification training parameters recommended by NVIDIA [33]. The model quantization process is conducted on 8 GeForce RTX 2080 Ti GPUs with CUDA 10.2 and PyTorch 1.5 frameworks on the Ubuntu 18.04 operating system.

The quantized models are then evaluated on two embedded FPGA platforms, Xilinx PYNQ-Z2 (i.e., XC7Z020) and ZCU102, to demonstrate the generality of our framework to different types of FPGA boards. The PYNQ-Z2 board contains only 220 DSPs and 53.2k LUTs, while the ZCU102 board has 2520 DSPs and 274.1k LUTs. To maximize the computation efficiency of the designs without timing violation, the working frequency of PYNQ-Z2 is set to 100 MHz for all DNN models, and that of ZCU102 is 150 MHz for all models. The data packing size is set as $G = 6$ on PYNQ-Z2 and $G = 8$ on ZCU102. The hardware designs are implemented through high-level synthesis of Xilinx Vivado 2020.1.

4.2 Accuracy Results

To justify the empirical findings that we mentioned in Section 3.1, we conduct verification experiments on ResNet-18 as shown in Table 2 and Table 3. The accuracy of unquantized baseline model from the TorchVision library [11] is listed in the first row in the table. Since the training parameters that we use may be different with the officially provided model, we fine-tune it with full precision to use as a comparison point shown in the last row, which can achieve 70.78% and 89.94% in top-1 and top-5 accuracy, respectively.

4.2.1 Accuracy Results under Different Mixed Ratio R. In Table 2, we compare the model accuracy under different mixed ratio R with a constant activation bit-width of 5-bit. First, when $R = 0\%$, i.e., all weights use 4-bit, the top-1 and top-5 model accuracy are 69.63% and 89.28%, respectively. That is, 4-bit quantization results in more than 1% top-1 accuracy drop. Second, when R reaches 100%, i.e., all weights use 8-bit, the top-1/top-5 accuracy is raised up to 70.60%/89.83%, which is nearly the same with the fine-tuned full precision model. Third, when R reaches 3%, 5%, 10%, and 20%, the model top-1 accuracy achieves 70.01%, 70.47%, 70.45%, and 70.56%, respectively, which indicates that $R = 5\%$ can efficiently reap the benefits from both higher accuracy and less weight bits. Increasing R to a value higher than 5% does not achieve an obvious accuracy improvement. Here we use the same $R = 5\%$ value across all layers, which already achieves a very good accuracy as shown in Table 2 and is friendly to design a single hardware accelerator that can be reused by all layers. Tuning a different R value for different layers may achieve a slightly better accuracy, but will lead to inefficient hardware implementation that is hard for reuse across layers.

Table 2: Comparisons of ResNet-18 model accuracy under different percentages of 8-bit weights on ImageNet dataset

Quantization Method	Weight Bit-width	Activation Bit-width	Top-1 Accuracy	Top-5 Accuracy
Baseline	32b	32b	69.57%	89.24%
Retrain	32b	32b	70.78%	89.94%
W4A5	4b	5b	69.63%	89.28%
Mixed	97% 4b + 3% 8b	5b	70.01%	89.33%
Mixed	95% 4b + 5% 8b	5b	70.47%	89.63%
Mixed	90% 4b + 10% 8b	5b	70.45%	89.55%
Mixed	80% 4b + 20% 8b	5b	70.56%	89.59%
Mixed (Inter)	4b + 8b	5b	69.92%	89.37%
W8A5	8b	5b	70.60%	89.83%

Table 3: Comparisons of ResNet-18 model accuracy under different bit-widths of activations on ImageNet dataset

Quantization Method	Weight Bit-width	Activation Bit-width	Top-1 Accuracy	Top-5 Accuracy
Baseline	32b	32b	69.57%	89.24%
Retrain	32b	32b	70.78%	89.94%
Mixed	95% 4b + 5% 8b	3b	68.91%	89.10%
Mixed	95% 4b + 5% 8b	4b	69.98%	89.38%
Mixed	95% 4b + 5% 8b	5b	70.47%	89.63%
Mixed	95% 4b + 5% 8b	6b	70.51%	89.64%

4.2.2 Accuracy Results under Different Activation Bit-widths. We explore the influence of activation bit-width as shown in Table 3. With constant mixed weight precision $R = 5\%$, we test the model accuracy under different activation bit-widths from 3-bit to 6-bit. The result shows that the model accuracy improves as the activation bit-width increases, while the higher the activation bit-width we have, the slower the improvement we obtain on the accuracy. When activation bit-width reaches 5-bit, the top-1/top-5 accuracy achieves 70.47%/89.63%, nearly the same as that of the 6-bit activation result. Therefore, we choose 5-bit as the activation bit-width in our FPGA experiments, which is more hardware efficient than 6-bit activations.

Here, we just use the verification experiments to demonstrate "5-bit activations and $R = 5\%$ 8-bit weights mixed with 4-bit weight" is an efficient combination. It does not mean that this one is the best to all kinds of models. It is worth noting that our framework is general to all kinds of weight/activation bit-widths. The change of the bit-width combination does not affect the framework availability.

4.3 Overall Throughput and Resource Results

To better understand the performance of FILM-QNN, in Table 4, we compare its peak throughput and overall resource utilization on

Table 4: Resource utilization and optimized computation throughput under different weight quantization precisions

Quantization Precision	Computation Resource	Implementation on ZCU102 (150 MHz)							Implementation on PYNQ-Z2 (100 MHz)						
		Utilization		W4A5 Op[a]		W8A5 Op		Peak Thrpt. (GOPS)	Utilization		W4A5 Op		W8A5 Op		Peak Thrpt. (GOPS)
		LUT	DSP	LUT	DSP	LUT	DSP		LUT	DSP	LUT	DSP	LUT	DSP	
100% W4	LUT	78%	1%	10240	0	-	-	1536	81%	13%	1728	0	-	-	172.8
	DSP	51%	82%	0	16384	-	-	2458	56%	95%	0	1440	-	-	144
	LUT + DSP	68%	78%	2048	15360	-	-	2611	77%	95%	576	1440	-	-	201.6
100% W8	LUT	76%	1%	-	-	6656	0	998.4	75%	13%	-	-	1152	0	115.2
	DSP	23%	82%	-	-	0	8192	1229	31%	95%	-	-	0	720	72
	LUT + DSP	65%	83%	-	-	3072	8192	1690	75%	95%	-	-	576	720	129.6
95% W4 + 5% W8	LUT	76%	2%	8192	0	1024	0	1382	81%	13%	1584	0	144	0	172.8
	DSP	54%	83%	0	14336	0	1024	2304	49%	95%	0	1152	0	144	129.6
	LUT + DSP	66%	83%	0	16384	1024	0	2611	78%	95%	720	1152	0	144	201.6

[a]The number of operations per cycle for 4-bit weights (W4) and 5-bit activations (A5). W8 means 8-bit weights.

Table 5: Comparisons of DNN implementations between previous studies, inter-layer quantization, and intra-layer quantization for ImageNet dataset on PYNQ-Z2 (XC7Z020) FPGA

Implementation	VGG [15]	ResNet-18 [2]	MobileNet-V2 [2]	ResNet-18 (Inter-Layer)	ResNet-50 (Inter-Layer)	MobileNet-V2 (Inter-Layer)	ResNet-18 (**Intra-Layer**)	ResNet-50 (**Intra-Layer**)	MobileNet-V2 (**Intra-Layer**)
Bit-Width	W8A8	W4A4		Middle W4A5, First & Last W8A5			95% W4A5 + 5% W8A5		
Top-1 Accuracy	67.62%	70.27%	65.64%	69.92%	77.08%	64.38%	70.47%	77.25%	65.67%
Frequency (MHz)	214	100		100			100		
kLUT	29.9 (56%)	28.3 (53%)		39.1 (74%)			41.3 (78%)		
DSP	190 (86%)	220 (100%)		214 (97%)			208 (95%)		
BRAM36	85.5 (61%)	56 (40%)		126.5 (90%)			123 (88%)		
Power (W)	3.5	-	-	3.0			3.5		
Frame Rate (FPS)	2.72	21.3	120.7	12.9	7.8	49.2	27.8	13.3	132.3
Throughput (GOPS)	84.3 (CONV)	77.0	71.8	46.8	63.6	29.3	100.6	108.6	78.7
GOPS/kLUT	2.825	2.725	2.538	1.197	1.627	0.749	2.436	2.629	1.907
GOPS/DSP	0.444	0.350	0.326	0.219	0.297	0.137	0.484	0.522	0.379
Energy Efficiency (GOPS/W)	24.1	-	-	15.6	21.2	9.8	28.7	31.0	22.5

PYNQ-Z2 and ZCU102 FPGA boards, under different quantization precisions and resource allocation strategies. Note that based on our modeling, FILM-QNN is not bounded by the on-chip and off-chip memory capacity and bandwidth (also confirmed by the resource utilization in Table 5), thanks to the data packing optimization presented in Section 3.2.3. Therfore, we mainly consider the most computation intensive operations in DNNs, namely the multiply-accumulate (MAC) operations, each equivalent to two operations. For each quantization precision (4-bit only, 8-bit only, and 95% 4-bit and 5% 8-bit for the weights), we explore the computing capability when 1) only using LUTs for MAC operations, 2) only using DSPs for MAC operations, and 3) using both LUTs and DSPs.

The achievable parallelism degree on each type of FPGA computation resources, i.e., the number of operations processed by LUTs or DSPs per cycle, is listed respectively for the operations with 4-bit weights (W4A5 Op columns) and 8-bit weights (W8A5 Op columns). Based on the total parallelism degree and working frequency of each implementation, the peak throughput is calculated in Giga Operations Per Second (GOPS). In the optimized implementations with 95% 4-bit and 5% 8-bit weight quantization, the peak throughput could achieve 2611 GOPS on ZCU102 and 201.6 GOPS on PYNQ-Z2.

Within the 95% 4-bit and 5% 8-bit weight quantization, using both LUTs and DSPs achieves 1.89× and 1.13× higher throughput than using LUTs only and using DSPs only on ZCU102, respectively. On PYNQ-Z2, these speedups are 1.17× and 1.56×, respectively. Similarly, within the 4-bit only and 8-bit only weight quantization precision, using both LUTs and DSPs achieves better throughput than using only LUTs or DSPs. This is because both the LUT and DSP utilization ratios in the mixed-precision implementations are high and more balanced than those in the designs with only LUTs or only DSPs for computations.

In general, we observe that it is better to first make full use of the DSP resources on FPGAs to accommodate as many MAC operations as possible, and then leverage the remaining LUTs to process more MAC operations. Note that when using DSPs for MAC operations, LUTs are also consumed due to data packing, and the LUT consumption corresponding to each DSP for W4A5 operations is about 2× higher than that for W8A5 operations, as one DSP carries out four W4A5 multiplications or two W8A5 multiplications.

Among different quantization precisions using both LUTs and DSPs, our 95% 4-bit and 5% 8-bit weight quantization achieves the highest peak throughput, which is the same as that of the 4-bit only weight quantization on both PYNQ-Z2 and ZCU102. Compared with the 8-bit only weight quantization, our mixed-precision weight quantization achieves 1.56× higher throughput on PYNQ-Z2 and 1.54× higher throughput on ZCU102. The 4-bit only weight quantization attains the same throughput as our mixed-precision

Table 6: Comparisons of DNN implementations between inter-layer and intra-layer quantization for ImageNet dataset on ZCU102 FPGA

Implementation	ResNet-18	ResNet-50	MobileNet-V2	ResNet-18	ResNet-50	MobileNet-V2
	(Inter-Layer)			(**Intra-Layer**)		
Bit-Width	Middle W4A5, First & Last W8A5			95% W4A5 + 5% W8A5		
Top-1 Accuracy	69.92%	77.08%	64.38%	70.47%	77.25%	65.67%
Frequency (MHz)	150			150		
kLUT	174.5 (64%)			180.1 (66%)		
DSP	2096 (83%)			2092 (83%)		
BRAM36	439 (48%)			440.5 (48%)		
Power (W)	13.4			12.9		
Frame Rate (FPS)	72.8	47.4	190.1	214.8	109.1	537.9
Thrpt. (GOPS)	263.7	387.8	113.3	778.9	891.4	320.1
GOPS/kLUT	1.511	2.222	0.649	4.324	4.948	1.777
GOPS/DSP	0.126	0.185	0.054	0.372	0.426	0.153
Energy Efficiency (GOPS/W)	19.7	28.9	8.5	60.4	69.1	24.8

quantization since the FPGA board could not accommodate more computations due to the routing issue.

4.4 Comparison with Prior Studies and Inter-Layer Quantization

Our implementations of FILM-QNN are further compared with two recent studies, i.e., Angel-Eye [15] with 8-bit weight and 8-bit activation quantization, and Mix-and-Match [2] with intra-layer mixed-scheme quantization (fixed-point and power-of-two quantization for 4-bit weight and 4-bit activation). Both studies use the PYNQ-Z2 (XC7Z020) FPGA platform. Table 5 shows their the top-1 accuracy, overall throughput (GOPS), frames per second (FPS), as well as the throughput per DSP and per thousands of LUTs (kLUT).

In terms of the model top-1 accuracy, FILM-QNN performs slightly better than Mix-and-Match [2] for both ResNet-18 and MobileNet-V2. Additionally, the top-1 accuracy of ResNet-18 and ResNet-50 achieved by FILM-QNN is much higher than that of VGG in Angel-eye [15] with 8-bit quantization. As for the computation throughput and frame rate, FILM-QNN achieves the best among the implementations on the PYNQ-Z2 board, namely 1.31× higher for ResNet-18 and 1.10× higher for MobileNet-V2 in throughput (and FPS) compared to Mix-and-Match [2]. As for resource utilization, since we prioritize the usage of DSPs, the throughput per DSP in FILM-QNN for ResNet-18 and MobileNet-V2 outperforms that of Mix-and-Match [2] on PYNQ-Z2. The throughput per thousands of LUTs in FILM-QNN is relatively lower, since a significant portion of LUTs are used to support the DSP computations (due to data packing) instead of performing the actual multiply-accumulate operations. In addition, the resource utilization results also confirm the high utilization of both DSPs and LUTs in FILM-QNN.

The throughput improvement over Mix-and-Match [2] for the MobileNet-V2 implementation is lower than that for the ResNet-18 implementation due to the small model size and depthwise convolution layers of MobileNet-V2. The overall parallelism degree for convolutional layers in our hardware implementations is mainly achieved by unrolling the filter and input channel dimensions. In MobileNet-V2, 31.5% of the layers have less than 128 filters, which does not provide enough parallelism along the filter dimension. Moreover, 31.5% of the layers apply only depthwise convolution operations, which eliminates the parallelism opportunity along the input channel dimension. In future work, we plan to further optimize our accelerator design to exploit more parallelism along other dimensions for such networks, which is orthogonal to demonstrate the benefits of our novel, accurate, and hardware-friendly intra-layer, mixed-precision quantization method.

We also compare between inter-layer and intra-layer quantization methods in Table 5 and Table 6. For the inter-layer quantization design, the computation resources are simultaneously allocated to W8A5 precision for the first and last network layers and W4A5 for other layers in a balanced way; however, they are not executing concurrently. The throughput increase from inter-layer to intra-layer quantization is 1.71× ∼ 2.69× on PYNQ-Z2, and 2.30× ∼ 2.95× on ZCU102. Another strategy for the inter-layer quantization design is to implement two FPGA bistreams respectively for pure W8A5 and pure W4A5 designs, and switch from W8A5 to W4A5 after the first layer and then from W4A5 to W8A5 before the last layer. Given that the bitstream reconfiguration time is about 33ms for PYNQ-Z2 and 213ms for ZCU102, the throughput improvement of intra-layer quantization over this inter-layer quantization design reaches 1.85× ∼ 9.78× on PYNQ-Z2, and 47.6× ∼ 230.2× on ZCU102. Finally, our intra-layer quantization also achieves 0.17% ∼ 1.29% higher accuracy over the inter-layer quantization.

In summary, our optimized intra-layer, mixed-precision implementations achieve comparable accuracy to the pure 8-bit precision designs, namely 70.47%, 77.25%, and 65.67% for ResNet-18, ResNet-50, and MobileNet-V2, respectively, and comparable throughput to the pure 4-bit precision designs, namely 27.8 FPS, 13.3 FPS, and 132.3 FPS on PYNQ-Z2, and 214.8 FPS, 109.1 FPS, and 537.9 FPS on ZCU102, for the three DNNs.

5 CONCLUSION

In this paper, we have designed and implemented a general framework, called FILM-QNN, to quantize and accelerate DNNs on FPGAs. To preserve the accuracy and hardware parallelism, we propose to quantize each layer of a DNN with a majority of weight filters as 4-bit and a minority of weight filters as 8-bit. In our FPGA accelerator design, we apply optimization techniques for low-precision computations, including DSP packing, weight reordering, and data packing. Moreover, we build a comprehensive model to guide the balanced allocation of FPGA resources to enable the concurrent computations with different precisions. Finally, we have evaluated FILM-QNN on three DNN models, i.e., ResNet-18, ResNet-50, and MobileNet-V2, on two Xilinx FPGA platforms, PYNQ-Z2 and ZCU102, using Vivado HLS. Our optimized mixed-precision implementations achieve comparable accuracy (70.47%, 77.25%, and 65.67% for the three models) to the 8-bit precision designs and comparable throughput (214.8 FPS, 109.1 FPS, and 537.9 FPS on ZCU102) to the 4-bit precision designs.

ACKNOWLEDGMENTS

This work is partly supported by NSF CCF-1901378; NSERC Discovery Grant RGPIN-2019-04613, DGECR-2019-00120, Alliance Grant ALLRP-552042-2020; CFI John R. Evans Leaders Fund.

REFERENCES

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

[2] Sung-En Chang, Yanyu Li, Mengshu Sun, Runbin Shi, Hayden K-H So, Xuehai Qian, Yanzhi Wang, and Xue Lin. 2021. Mix and Match: A novel FPGA-centric deep neural network quantization framework. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 208–220.

[3] Gong Cheng, Lu Ye, Li Tao, Zhang Xiaofan, Hao Cong, Chen Deming, and Chen Yao. 2019. μL2Q: An Ultra-Low Loss Quantization Method for DNN. *The 2019 International Joint Conference on Neural Networks (IJCNN)* (2019), 1–8.

[4] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).

[5] Philip Colangelo, Nasibeh Nasiri, Eriko Nurvitadhi, Asit Mishra, Martin Margala, and Kevin Nealis. 2018. Exploration of low numeric precision deep learning inference using intel® FPGAs. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 73–80.

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems (NeurIPS)*. 3123–3131.

[7] 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).

[8] Zhen Dong, Zhewei Yao, Yaohui Cai, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. HAWQ-V2: Hessian Aware trace-Weighted Quantization of neural networks. *arXiv preprint arXiv:1911.03852* (2019).

[9] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 293–302.

[10] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned step size quantization. *International Conference on Learning Representations (ICLR)* (2019).

[11] Facebook. 2021. Torchvision. https://pytorch.org/vision/stable/models.html Last accessed Sept 12, 2021.

[12] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 4852–4861.

[13] Google. 2021. TensorFlow. https://www.tensorflow.org/lite Last accessed May 27, 2021.

[14] K. Guo, W. Li, K. Zhong, Z. Zhu, S. Zeng, S. Han, Y. Xie, P. Debacker, M. Verhelst, and Y. Wang. 2021. Neural Network Accelerator Comparison. https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/.

[15] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 37, 1 (2017), 35–47.

[16] Peng Guo, Hong Ma, Ruizhi Chen, Pin Li, Shaolin Xie, and Donglin Wang. 2018. Fbna: A fully binarized neural network accelerator. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 51–513.

[17] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)* (2016).

[18] Zhezhi He and Deliang Fan. 2019. Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 11438–11446.

[19] Intel. 2017. Intel Arria 10 Native Fixed Point DSP IP Core User Guide. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_nfp_dsp.pdf Last accessed Sept 11, 2021.

[20] Li Jiao, Cheng Luo, Wei Cao, Xuegong Zhou, and Lingli Wang. 2017. Accelerating low bit-width convolutional neural networks with embedded FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–4.

[21] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[22] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. 2019. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4350–4359.

[23] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. 2018. Extremely low bit neural network: Squeeze the last bit out with admm. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*.

[24] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).

[25] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*. 345–353.

[26] Qian Lou, Feng Guo, Minje Kim, Lantao Liu, and Lei Jiang. 2019. AutoQ: Automated Kernel-Wise Neural Network Quantization. In *International Conference on Learning Representations (ICLR)*.

[27] Alec Lu, Zhenman Fang, Weihua Liu, and Lesley Shannon. 2021. Demystifying the memory system of modern datacenter FPGAs for software programmers through microbenchmarking. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 105–115.

[28] Cheng Luo, Wei Cao, Lingli Wang, and Philip HW Leong. 2019. Rna: An accurate residual network accelerator for quantized and reconstructed deep neural networks. *IEICE Transactions on Information and Systems* 102, 5 (2019), 1037–1045.

[29] Daisuke Miyashita, Edward H Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025* (2016).

[30] Hiroki Nakahara, Tomoya Fujii, and Shimpei Sato. 2017. A fully connected layer elimination for a binarized convolutional neural network on an FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–4.

[31] Hiroki Nakahara, Haruyoshi Yonekawa, Tsutomu Sasao, Hisashi Iwamoto, and Masato Motomura. 2016. A memory-based realization of a binarized deep convolutional neural network. In *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 277–280.

[32] Dong Nguyen, Daewoo Kim, and Jongeun Lee. 2017. Double MAC: Doubling the performance of convolutional neural networks on modern FPGAs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 890–893.

[33] Nvidia. 2021. Nvidia Deep Learning Examples. https://github.com/NVIDIA/DeepLearningExamples Last accessed Sept 12, 2021.

[34] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. 2018. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 580–595.

[35] 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), 2017*. IEEE, 1147–1152.

[36] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision (ECCV)*. Springer, 525–542.

[37] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. 2019. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 925–938.

[38] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Press, 764–775.

[39] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT.. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*. 8815–8821.

[40] Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. 2020. Mixed Precision DNNs: All you need is a good parametrization. *International Conference on Learning Representations (ICLR)* (2020).

[41] 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 *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 65–74.

[42] Mário Véstias, Rui Policarpo Duarte, José T de Sousa, and Horácio Neto. 2017. Parallel dot-products for deep learning on FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–4.

[43] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin, and Deming Chen. 2018. Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 163–1636.

[44] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), 8604–8612.

[45] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed precision quantization of convnets via differentiable

neural architecture search. *arXiv preprint arXiv:1812.00090* (2018).

[46] Xilinx. 2017. Deep Learning with INT8 Optimization on Xilinx Devices. https://www.xilinx.com/support/documentation/white_papers/wp486-deep-learning-int8.pdf Last accessed Sept 12, 2021.

[47] Xilinx. 2020. Convolutional Neural Network with INT4 Optimization on Xilinx Devices. https://www.xilinx.com/support/documentation/white_papers/wp521-4bit-optimization.pdf Last accessed Sept 12, 2021.

[48] Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, John Wawrzynek, and Kurt Keutzer. 2019. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 23–32.

[49] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael Mahoney. 2019. PyHessian: Neural networks through the lens of the Hessian. *arXiv preprint arXiv:1912.07145* (2019).

[50] 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 *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays (FPGA)*. 161–170.

[51] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*. 365–382.

[52] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2018. DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[53] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 15–24.

[54] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* (2017).

[55] 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).

[56] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2017. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*.