

# DeepBurning-MixQ: An Open Source Mixed-Precision Neural Network Accelerator Design Framework for FPGAs

Erjing Luo<sup>1,2\*</sup>, Haitong Huang<sup>1,3\*</sup>, Cheng Liu<sup>1†</sup>, Guoyu Li<sup>1,3</sup>, Bing Yang<sup>4</sup>, Ying Wang<sup>1</sup>, Huawei Li<sup>1</sup>, Xiaowei Li<sup>1</sup>  
<sup>1</sup>SKLP, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China  
<sup>2</sup>School of Information and Electronics, Beijing Institute of Technology, Beijing, China  
<sup>3</sup>Dept. of Computer Science, University of Chinese Academy of Sciences, Beijing, China  
<sup>4</sup>Dept. of Computer Science and Technology, Harbin University of Science of Technology, Harbin, China  
{huanghaitong21s, liucheng, liguoyu21s}@ict.ac.cn, 1120192664@bit.edu.cn

**Abstract**—Mixed-precision neural networks (MPNNs) that enable the use of just enough data width for a deep learning task promise significant advantages of both inference accuracy and computing overhead. FPGAs with fine-grained reconfiguration capability can adapt the processing with distinct data width and models, and hence, can theoretically unleash the potential of MPNNs. Nevertheless, commodity DPUs on FPGAs mostly emphasize generality and have limited support for MPNNs especially the ones with lower data width. In addition, primitive DSPs in FPGAs usually have much larger data width than that is required by MPNNs and haven't been sufficiently co-explored with MPNNs yet. To this end, we propose an open source MPNN accelerator design framework specifically tailored for FPGAs. In this framework, we have a systematic DSP-packing algorithm to pack multiple lower data width MACs in a single primitive DSP and enable efficient implementation of MPNNs. Meanwhile, we take DSP packing efficiency into consideration with MPNN quantization within a unified neural network architecture search (NAS) framework such that it can be aware of the DSP overhead during quantization and optimize the MPNN performance and accuracy concurrently. Finally, we have the optimized MPNN fine-tuned to a fully pipelined neural network accelerator template based on HLS and make best use of available resources for higher performance. Our experiments reveal the resulting accelerators produced by the proposed framework can achieve overwhelming advantages in terms of performance, resource utilization, and inference accuracy for MPNNs when compared with both handcrafted counterparts and prior hardware-aware neural network accelerators on FPGAs.

**Index Terms**—DSP packing, mixed precision neural network, neural network architecture search, quantization and implementation co-optimization.

## I. INTRODUCTION

Quantization is a straightforward yet effective approach [1] [2] to compress neural network models that are usually both computing- and memory-intensive. Since neural network's sensitivity to quantization varies across the layers, mixed-precision quantization that allows more fine-grained quantization achieves significant advantages in both computing efficiency and memory access efficiency compared to classical

uniform quantization [3] [4], which contributes to the neural network processing throughput and energy efficiency eventually. Nevertheless, most of the commodity neural network computing engines including CPU, GPU, and NPU usually have limited support for arbitrary mixed-precision neural network (MPNN) processing especially low data width processing due to the lack of native mixed precision computing elements. In contrast, FPGAs with fine-grained reconfiguration capability can provide model specific implementation [5] [6] and suit various computing requirements of MPNNs with native hardware, and hence, can unleash the potential of MPNNs.

Despite the fine-grained reconfiguration capability, FPGAs mainly rely on digital signal processing unit (DSP) cores with limited reconfiguration for efficient arithmetic implementations. For example, Xilinx integrates DSP48E2 which can support a  $27 \times 18$  two's complement multiplication on its UltraScale FPGAs [7], while Intel Arria 10 devices have DSP cores that can be configured to two  $18 \times 19$  multipliers or one  $27 \times 27$  multiplier [8]. Although Lookup-Tables (LUTs) can also be utilized to implement arithmetic operations with arbitrary data width, DSPs generally outperform the LUT-based implementations in terms of both latency and energy efficiency [9] especially for higher data width operations. While MPNNs typically involve many low data width operations that are much smaller than data width of primitive DSPs, straightforward implementation of MPNNs can result in considerable waste of the DSP resources. To address the problem, a variety of works have attempted to pack multiple low data width operations into a single DSP block [10] [11] [12] and make full use of the computing capability of the DSPs. For instance, Xilinx's INT8 [12] and INT4 [11] demonstrate that two 8-bit multiplications and four 4-bit multiplications can be fit into a single DSP48E2. The authors in [10] also explored the use of high data width processing engines for low data width processing.

While MPNN quantization affects not only the accuracy but also the DSP packing efficiency and performance eventually, performing the quantization and DSP packing separately will lead to sub optimal results. In fact, there have been intensive efforts devoted to optimize the neural network model accuracy and performance at the same time. For instance, [13] employs

\* These authors contributed equally to this work.

† Corresponding authors.

This work is supported by the National Key R&D Program of China under Grant (2022YFB4500405), and the National Natural Science Foundation of China under Grant 62174162.

a RL agent with direct hardware metrics feedback to co-optimize accuracy, latency, and energy consumption. [14] incorporates quantization and other neural architecture parameters into a design space, and has gradient-descent method for optimizing both algorithms and hardware implementation. However, there is still a lack of co-optimization between DSP packing and MPNN quantization.

In this work, we propose a systematic DSP packing algorithm that can squeeze multiple low data width arithmetic operations into a single primitive DSP of FPGAs. While the DSP packing efficiency varies substantially across the convolution kernels with different parameters, the overall performance of a MPNN network can be inconsistent with the data width of the model. Then, we leverage a differentiable NAS to take both the DSP packing and MPNN quantization into consideration and co-optimize the model accuracy and performance at the same time. Finally, with the optimized DSP packing and quantization determined by the NAS, we have a MPNN accelerator generated automatically based on a fully pipelined neural network accelerator template.

The major contributions of this work can be summarized as follows:

- We propose a mixed DSP packing algorithm that takes advantage of both *Kernel Packing* strategy and *Filter Packing* strategy for arbitrary low data width convolution on FPGAs. In addition, we also enhance the DSP packing with *Overpacking* technique and *Operation Separation* technique. The resulting DSP packing algorithm outperforms all the existing strategies significantly.
- On top of the proposed DSP packing algorithm, we propose a MPNN accelerator design framework that leverages a differentiable NAS to take both the DSP packing and quantization into a consideration and optimizes the model accuracy and performance at the same time. With the optimized DSP packing strategy and quantization setups, this framework can further generate high-performance MPNN accelerator based on a fully pipelined HLS template. The framework is open sourced on Github<sup>1</sup>.
- According to our experiments on a set of different MPNNs, the MPNN accelerators generated with the proposed framework outperform state-of-the-art counterparts in terms of performance, resource utilization, and prediction accuracy significantly.

## II. RELATED WORK

Mixed-precision neural networks (MPNNs) that enable just enough data width for each different neural network layer can greatly reduce the requirements of computing, memory bandwidth, and storage. Therefore, MPNNs promise great computing efficiency when compared to neural network models with unified data width. Since the number of low bit-width operations is inconsistent with the realistic computing efficiency due to the lack of primitive MPNN implementation on existing

computing engines including CPUs, GPUs, and NPU. Many prior work seek to co-optimize the quantization and computing efficiency [15] [16] to ensure efficient MPNN implementation on specific computing engines. Specifically, [13] [17] [18] [19] [20] [21] leverage network architecture search (NAS) technique that automates the neural network design for the co-optimization. For instance, [20] has a hardware performance model added to differentiable NAS such that performance of neural network candidates on GPUs can be evaluated with the model accuracy at the same time. Similar approaches have also been successfully applied to lightweight neural network design for mobile phones [22]. However, the above hardware aware neural network design and optimization approaches essentially adapt the neural network models to the target computing engines that have fixed computing architectures and have little native low data width operation support, and hence, fail to fully unleash the potential of MPNNs.

In contrast, FPGAs with fine-grained reconfiguration capability are suitable for model specific customization and can be a good fit for MPNNs. To explore the reconfiguration capability of FPGAs for efficient neural network specific implementation, the authors in [3] [13] [14] proposed different co-optimization approaches that take both neural network accuracy and hardware implementation efficiency into consideration in a unified framework. [13] applies time-consuming reinforcement learning NAS. Although such NAS approach could have the accelerator design parameters included in the same NAS search space along with the network architecture and have different design metrics considered at the same time during NAS evaluation stage, it enlarges the search space dramatically and usually induce many time-consuming evaluation of metrics such as accuracy, hardware overhead, and implementation quality, which makes the entire optimization prohibitively expensive and difficult to converge. [14] formulates the co-optimization as a differentiable NAS problem in terms of both accuracy and implementation efficiency. It avoids the conventional iterative NAS search procedures and reduces the optimization to a standard training procedure. Essentially, it makes the hardware-aware optimization much easier to converge. However, it is difficult to explicitly represent and model all hardware design parameters within a differentiable NAS framework, as many hardware parameters are discrete, making it difficult to construct differentiable proxy loss and obtain global optimal solutions. Different from these above NAS approaches, [3] takes the co-optimization as a *integer programming* problem by introducing an accuracy predictor and a performance predictor, which do not rely on any complex black box simulation or evaluation procedures. Particularly, it demonstrates the great potential of using a simplified co-optimization framework for neural network acceleration on FPGAs, but the accuracy prediction can be relatively limited to some specific scenarios.

In addition, the fine-grained reconfiguration capability of FPGAs has not been explored sufficiently. FPGAs mainly rely on primitive DSP blocks with fixed data width for high performance arithmetic operations while straightforward

<sup>1</sup><https://github.com/fffasttime/AnyPackingNet/>

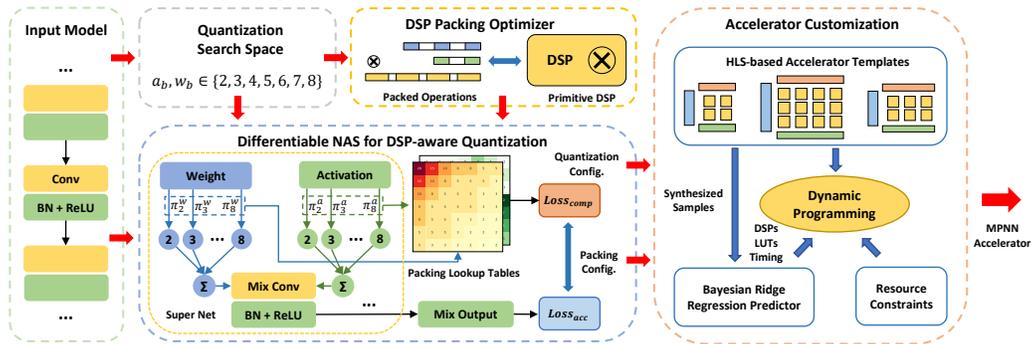


Fig. 1. Overview of the proposed mixed-precision neural network accelerator design framework.

implementation of low data width such as 2-bit and 3-bit operations on these primitive DSP blocks can lead to considerable waste because the data width of these DSP blocks is much larger. LUTs in FPGAs can also be utilized to construct operations with arbitrary data width, but the performance is usually much lower especially for operations with larger data width due to the more complex routing. To fully explore the fine-grained reconfiguration capability of FPGAs, researchers from both academia and industry have proposed a variety of approaches to pack multiple low-precision operations into a single DSP (especially multiplier) concurrently to fully utilize the primitive DSPs [9] [10] [11] [12] [23] and then extract the outputs of the low-precision operations from the disjoint bit segments of the DSP blocks concurrently. For instance, Xilinx INT4 optimization [11] leverages the  $27 \times 18$  two's complement multiplier to simultaneously calculate four 4-bit products. [24] [25] proposes to construct efficient low data width matrix-matrix multiplication overlay based on primitive DSP blocks. Particularly, it takes the interconnection between primitives into consideration for the sake of higher operation frequency. Then, it has neural network models implemented based on the overlay to ensure high-performance neural network processing on FPGAs. Essentially, it optimizes the hardware implementation first without being aware of the models and adapts the model to the FPGA implementation afterwards. HiKonv [10] specifically explores the DSP packing algorithm to maximize low data width operations on a single DSP block, but there is a lack of co-optimization of the hardware implementation and neural network models. In summary, there is still a lack of co-optimization framework that takes the neural network model accuracy and fine-grained FPGA implementation efficiency into consideration at the same time for FPGAs.

### III. MPNN ACCELERATOR DESIGN FRAMEWORK

In this work, we propose a mixed-precision neural network accelerator design framework as shown in Fig. 1. Essentially, it is a hardware and software co-optimization framework for MPNNs and seeks to optimize both the processing performance and accuracy. In general, it adjusts the quantization of MPNNs to fulfill the accuracy requirement and optimize the resulting accelerator implementation which mainly relies on the various low data width operation mapping efficiency over primitive DSPs within FPGAs.

The framework starts with a floating point neural network model or fixed point model with unified quantization. With the model architecture, it utilizes differentiable NAS to determine the optimized MPNN quantization. Specifically, it defines the quantization search space in which the data width of the model in each layer ranges from 2bit to 8bit. Based on the search space, it further extends the input neural network model by adding all the possible candidate quantization branches to all the links between layers in the original input neural network and constructs a super-net for the NAS. The branches in the super-net are weighted and they can be adapted to optimize the model accuracy through back propagation. Other than the accuracy, it also has the NAS to be aware of the hardware implementation efficiency especially the DSP requirements which is usually the resource bottleneck. While the hardware implementation relies on the model quantization in each layer as well as the low data width operation mapping efficiency over primitive DSPs in FPGAs, a *DSP Packing Optimizer* is utilized to pack the various low data width operations of MPNNs within primitive DSPs. Then, we have the optimal DSP packing configurations produced by *DSP Packing Optimizer* stored in a lookup table such that they can be referred to immediately during NAS and utilized to co-optimize the MPNN accuracy and DSP overhead. Basically, we utilize a combined accuracy and DSP overhead loss to train the super-net where the accuracy loss is obtained through standard forward processing and DSP overhead is evaluated based on the lookup table of the DSP packing. At the end of the super-net training, the branches with highest weights will be selected as the optimized quantization configurations and DSP packing configurations accordingly.

After the DSP-aware quantization, the framework proceeds to the accelerator customization stage. Essentially, it orchestrates the design parameters of our pipelined neural network accelerator templates based on the optimized quantization and DSP packing configurations of the neural network model for the sake of higher performance under the specified FPGA resource constraints. Essentially, it is an resource allocation problem that allocates the hardware resources to different pipeline stages such that the implementation of each pipeline stage is optimized and the performance of the different pipeline stages are balanced at the same time. To address the constrained resource allocation problem, we have a *dynamic*

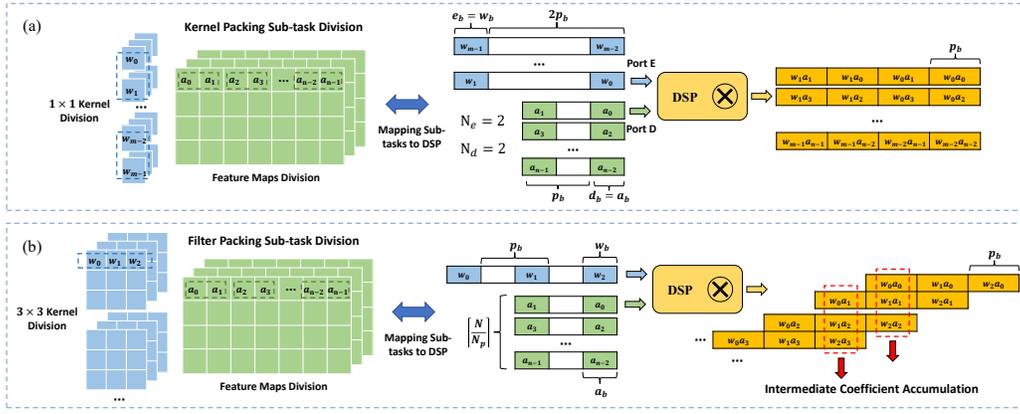


Fig. 2. DSP packing strategies: (a) Kernel Packing; (b) Filter Packing. Suppose both weights and activations are unsigned.

programming algorithm to tune the design parameters of the pipelined accelerator. The tuning procedure requires a large number of evaluation of different design options and the evaluation metrics can be hardware overhead like DSPs and timing quality, which are prohibitively expensive using standard tools from FPGA vendors. In this work, we utilize a *Bayesian Ridge Regression* predictor to estimate resource utilization and the timing of each pipeline stage implementation. Although it needs additional sampling data for pre-training of the predictors, the resulting prediction models can be orders of magnitude faster and ensures rapid accelerator customization of a specific MPNN.

#### IV. DSP PACKING OPTIMIZER

In order to efficiently map mixed-precision arithmetic operations onto primitive DSPs in FPGAs, we propose our *DSP Packing Optimizer* which further generalizes the state-of-the-art DSP packing algorithms [10] [11] [12] as two optional strategies and also incorporates two additional techniques for further enhancement. For a convolution operator, the optimizer traverses all possible packing configurations to find the optimal one for each bit-width combination, and stores it in lookup tables to direct quantization search and hardware customization.

##### A. DSP Packing Strategies

1) *Kernel Packing*: For the first strategy, as illustrated in Fig. 2(a), weights and activations respectively from adjacent kernels and pixels are squeezed into one DSP following Eq. 1. Specifically, through left-shift and addition,  $N_d$   $d_b$ -bit weights (activations) and  $N_e$   $e_b$ -bit activations (weights) are mapped onto two input ports, D and E, where we assume port E's bit-width is larger or equal to port D's (i.e.  $P_b^E \geq P_b^D$ ), then  $N_d \times N_e$  independent multiplications are concurrently performed within one DSP. The  $g_b$  in Eq. 1 is called guard bits, which are deliberately preserved to prevent bit segment overlap between neighbor multiplications or to support overall result accumulation for saving decoding logic.

$$\left( \sum_{i=0}^{N_d-1} d[i]2^{ip_b} \right) \cdot \left( \sum_{j=0}^{N_e-1} e[j]2^{jN_dp_b} \right) \quad (1)$$

subject to

$$\begin{cases} p_b = d_b + e_b + g_b \\ g_b \geq 0 \\ d_b + (N_d - 1)p_b \leq P_b^D \\ e_b + (N_e - 1)N_dp_b \leq P_b^E \end{cases}$$

2) *Filter Packing*: Instead of packing weights from adjacent kernels, *Filter Packing* factorizes multidimensional convolution into 1-D counterparts, and packs the filter on an input port. Suppose there are a  $K_p$ -element weight filter  $f$  and a  $N_p$ -element activation sequence  $s$ . Based on the mathematical equivalence between 1-D convolution and polynomial multiplication, the convolution ( $c = f * s$ ) can be reformulated as polynomial representation, as shown in Eq. 2, which can then be implemented with one large bit-width multiplier. However, in practice, due to the limitation of port widths (i.e.  $P_b^A$  and  $P_b^W$ ), one multiplier cannot contain a large convolution entirely. Therefore, for processing a long  $K$ -element filter and a long  $N$ -element sequence, this strategy can be generalized by dividing the original polynomial into  $\lceil \frac{K}{K_p} \rceil \times \lceil \frac{N}{N_p} \rceil$  sub-tasks. As illustrated in Fig. 2(b), through iteratively calculating these sub-tasks and accordingly accumulating intermediate coefficients, correct convolution can be obtained. In this case, the guard bits must be greater than  $\lceil \log_2 \min\{K_p, N_p\} \rceil$ , since  $\min\{K_p, N_p\}$  accumulations are inherently introduced by its polynomial nature.

$$\begin{aligned} c(2^{p_b}) &= f(2^{p_b})s(2^{p_b}) = \left( \sum_{i=0}^{K_p-1} f[i]2^{ip_b} \right) \cdot \left( \sum_{j=0}^{N_p-1} s[j]2^{jp_b} \right) \\ &= \sum_{i=0}^{K_p+N_p-1} c[i]2^{ip_b} \end{aligned} \quad (2)$$

subject to

$$\begin{cases} p_b = a_b + w_b + g_b \\ g_b \geq \lceil \log_2 \min\{K_p, N_p\} \rceil \\ a_b + (N_p - 1)p_b \leq P_b^A \\ w_b + (K_p - 1)p_b \leq P_b^W \end{cases}$$

3) *Mixed Packing*: Compared with *Kernel Packing* that leaves larger space (i.e.  $N_dp_b$ ) between the operands on port

E to guarantee multiplications' independence, *Filter Packing* can pack operands more densely. However, for small kernel size, this advantage might not be unleashed as the filter cannot fully occupy DSP's port width (e.g. point-wise convolution). For fairly comparing different strategies and configurations, we use two metrics to evaluate our DSP packing strategies.

Primarily, we expect to maximally accommodate multiplications into these DSP primitives for highest multiplication throughput. However, we do not intuitively define it as the total number of concurrent multiplications in one DSP, because this neglects the up-rounding redundancy introduced by sub-task division in *Filter Packing*, especially the division for filter whose length is shorter and can lead to severe waste. Hence, we define the multiplication throughput  $T_{mul}$  as the average of the effective multiplication performed in one DSP, as shown in Eq. 3. Next, based on the consideration that guard bits can be utilized for supporting accumulations before decoding, we also expect our optimizer to increase it. As *Filter Packing* inherently introduces accumulations, we define extra guard bits  $E_g$  with Eq. 4, and regard it as the subsidiary optimization objective.

$$T_{mul} = \begin{cases} N_d N_e, & \text{Kernel Packing} \\ \frac{KN}{\lceil \frac{K}{K_p} \rceil \lceil \frac{N}{N_p} \rceil}, & \text{Filter Packing} \end{cases} \quad (3)$$

$$E_g = \begin{cases} g_b, & \text{Kernel Packing} \\ g_b - \lceil \log_2 \min\{K_p, N_p\} \rceil, & \text{Filter Packing} \end{cases} \quad (4)$$

### B. Packing Enhancement

As explained in Eq. 1 and Eq. 2, multiplication throughput is restricted by both guard bits and port width constrains. In order to further enhance it, our optimizer also introduces two enhancement techniques.

1) *1-bit Overpacking*: The guard bits in packing algorithms are deliberately preserved to avoid result overlap. Nonetheless, they inevitably consume the precious bit-widths. Inspired by [9], we introduce *1-bit overpacking* (i.e. allowing 1-bit overlap) to mitigate this constrain, and further propose a method to fully compensate the error.

When 1-bit overlap is permitted, the LSB of high-position segment  $B_{LS}^H$  and the MSB of low-position segment  $B_{MS}^L$  overlaps with each other. In order to decode correctly, we recalculate  $B_{LS}^H$  with additional LUT resources, and leverage it for calibration. As shown in Fig. 3, the LSB of a product can be obtained by applying an AND gate to the LSBs of its multiplicands. If the high-position segment is the sum of several products, we apply an XOR logic to all products' LSBs to calculate  $B_{LS}^H$ . For correcting  $B_{MS}^L$ , we directly add the recalculated  $B_{LS}^H$  to it to counteract the contamination. For high-position segment, the error is possibly introduced by the sign extension of the low-position segment, which is equivalent to subtracting the result by one. To detect and compensate the unknown extension, we add the XOR of the overlapped bit and  $B_{LS}^H$  for correction.

2) *Operand Separation*: In packing algorithms, we expect to find suitable  $K_p$  and  $N_p$  (or  $N_d$  and  $N_e$ ) combinations that can fully occupy the precious port widths. However, as these

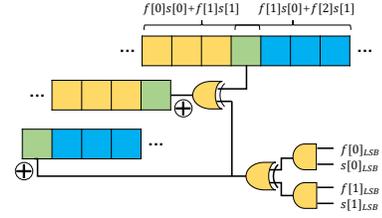


Fig. 3. 1-bit overpacking correction.

values are discrete, it's unavoidable to leave some bit-width unused. This phenomenon is especially conspicuous for large bit-width combinations, since the available packing choices are more limited. Therefore, we propose using *Operand Separation* to alleviate it.

The basic idea is to split a high bit-width operand (weight or activation) into two low bit-width counterparts. Take *Filter Packing* as an example. Following Eq. 5, a  $w_b$ -bit filter  $f[k]$  can be split into a  $(w_b - \lceil w_b/2 \rceil)$ -bit  $f_H[k]$  and a  $(\lceil w_b/2 \rceil)$ -bit  $f_L[k]$ . Then, the original polynomial multiplication can be reformulated as two low bit-width counterparts, and be separately implemented with two multipliers. Clearly, the separation halves the number of concurrent multiplications in one DSP. However, the resulting smaller bit-widths might be able to occupy input ports more densely, thus might serve to further enhance multiplication throughput.

$$\begin{aligned} c(2^{p_b}) &= \sum_{i=0}^{K_p-1} \sum_{j=0}^{N_p-1} f[i]s[j]2^{(i+j)p_b} \\ &= \sum_{i=0}^{K_p-1} \sum_{j=0}^{N_p-1} (f_H[k]2^{\lceil \frac{w_b}{2} \rceil} + f_L)s[j]2^{(i+j)p_b} \\ &= 2^{\lceil \frac{w_b}{2} \rceil} f_H(2^{p_b})s(2^{p_b}) + f_L(2^{p_b})s(2^{p_b}) \end{aligned} \quad (5)$$

### V. DSP-AWARE QUANTIZATION

Inspired by prior works [14] [19], we mainly leverage the differentiable NAS technique for hardware-efficient quantization of MPNNs. As DSP is usually the resource bottleneck of NN accelerators on FPGAs [5] [9] [14], it is utilized as the major hardware metric in the hardware-aware NAS.

Given input model  $\mathcal{A}$ , a super-net with candidate quantization branches as shown in Fig. 1 will be generated. Each branch is assigned an architecture parameter  $\pi_i$  to represent selection probability. The accuracy and hardware evaluation metrics are then formulated as differentiable functions with respect to these parameters. For inference accuracy, each layer's weight and activation candidate branches are weighted by these selection parameters in propagation such that the mixed accuracy loss  $Loss_{acc}(\pi^w, \pi^a)$  is calculated. For hardware-efficiency, to fully exploit the critical DSP resources on FPGAs, we regard the packed multiplications as one DSP operation, and propose to use the total DSP operations of all layers for evaluation, as defined in Eq. 6, where  $Op_{mul}^l$ ,  $Q_w^l$ , and  $Q_a^l$  denote the number of multiplication operations and quantization bit-widths in the  $l^{th}$  layer respectively. To orchestrate it with the super-net, each layer's multiplication

throughput lookup table  $T_{mul}^l$  is weighted with the corresponding selection probability as shown in Eq. 7. The complexity loss  $Loss_{comp}(\pi^w, \pi^a)$  is therewith defined as the probability expectation of the total DSP operations, and is added to accuracy loss with an adjustable hyper-parameter  $\eta$  for tuning relative significance as shown in Eq. 9. In back-propagation, the two loss functions are minimized based on target dataset until converge or reaching certain epochs, through which the bit-width selections are consequently optimized. In the end, the sub-path with highest selection probability is chosen as the finalized quantization configurations.

$$Op_{dsp} = \sum_{l=1}^L \frac{Op_{mul}^l}{T_{mul}^l(w_b^l, a_b^l)} \quad (6)$$

$$\overline{T_{mul}^l}(\pi^w, \pi^a) = \sum_{w_b \in Q_w} \sum_{a_b \in Q_a} \pi_i^w \pi_j^a T_{mul}^l(w_b, a_b) \quad (7)$$

$$Loss_{comp}(\pi^w, \pi^a) = \sum_{l=1}^L \frac{Op_{mul}^l}{\overline{T_{mul}^l}(\pi^w, \pi^a)} \quad (8)$$

$$Loss(\pi^w, \pi^a) = Loss_{acc}(\pi^w, \pi^a) + \eta Loss_{comp}(\pi^w, \pi^a) \quad (9)$$

## VI. ACCELERATOR CUSTOMIZATION

We use HLS to design hardware templates based on our DSP packing algorithms. After quantization search, the framework will automatically configure the templates and map each layer as a pipeline stage. Each stage will be assigned  $P_{fl}^l$  DSPs to construct a parallel computing array similar with [5]. However, as LUTs can also be efficient computing resources when bit-width is small [3] [14], our design also provides an alternative to construct computing arrays with equivalent LUT arithmetic, but it must satisfy timing constrain. Given the maximum DSPs  $R_{dsp}^{max}$  and LUTs  $R_{lut}^{max}$ , we formulate the accelerator customization problem as Eq. 10. Here, as pipeline stages are connected through FIFOs, we assume the overall Worst Negative Slack (WNS) is determined by the worst stage.

To solve above problem, we randomly sample and synthesize a set of possible hardware configurations with our templates, and pre-train a *Bayesian Ridge Regression* model to estimate each stage's DSPs  $\hat{R}_{dsp}^l$ , LUTs  $\hat{R}_{lut}^l$ , and WNS  $\hat{t}_{wns}^l$ . Next, we propose utilizing *dynamic programming* to find the optimal resource allocation, as shown in Algorithm 1. It employs a three-dimension table to memorize the optimal pipeline configurations of a sub-problem, and leverages recurrence relation to solve a larger sub-problem. Specifically, we use  $Lat[l][R_{dsp}^{cur}][R_{lut}^{cur}]$  to represent the minimal latency when  $R_{dsp}^{cur}$  DSPs and  $R_{lut}^{cur}$  LUTs are available for the first  $l$ -stage pipeline of the entire design. If  $\hat{R}_{dsp}^l$  out of the  $R_{dsp}^{cur}$  DSPs and  $\hat{R}_{lut}^l$  out of the  $R_{lut}^{cur}$  LUTs are allocated for the  $l^{th}$  stage, the recurrence relation can be formulated as Eq. 11. Based on this, we can search for the optimal solution in bottom-up fashion. The time complexity of the proposed algorithm only grows linearly with the depth and resource scale, which means a thorough design space exploration is completely affordable.

---

### Algorithm 1: Accelerator Customization

---

```

1 Initialize all  $Lat = \max\{Op_{dsp}^l\}$  and Config.
2 for  $l = 1, 2, \dots, L$  do
3   for  $R_{dsp}^{cur} = 0$  to  $R_{dsp}^{max}$  do
4     for  $R_{lut}^{cur} = 0$  to  $R_{lut}^{max}$  do
5       for all possible  $P_{fl}^l$  do
6         Estimate  $\hat{R}_{dsp}^l, \hat{R}_{lut}^l$ , and  $\hat{t}_{wns}^l$ 
7          $Lat^{new} = \max\{\frac{Op_{dsp}^l}{P_{fl}^l}, Lat^{pre}\}$ 
8          $C1 = Lat^{new} < Lat[l][R_{dsp}^{cur}][R_{lut}^{cur}]$ 
9          $C2 = \hat{R}_{dsp}^l \leq R_{dsp}^{cur} \ \& \ \hat{R}_{lut}^l \leq R_{lut}^{cur}$ 
10         $C3 = \hat{t}_{wns}^l > 0$ 
11        if  $C1 \& C2 \& C3$  then
12          update  $Lat[l][R_{dsp}^{cur}][R_{lut}^{cur}]$  and
            corresponding Config.
13 Return  $Lat[L][R_{dsp}^{max}][R_{lut}^{max}]$  and Config.
```

---

$$\begin{aligned} \min \quad & Lat = \max\{Lat^l\} = \max\left\{\frac{Op_{dsp}^l}{P_{fl}^l}\right\} \\ \text{s.t.} \quad & \sum_{l=1}^L R_{dsp}^l \leq R_{dsp}^{max}, \quad \sum_{l=1}^L R_{lut}^l \leq R_{lut}^{max} \\ & \min\{t_{wns}^l\} > 0 \end{aligned} \quad (10)$$

$$Lat[l][R_{dsp}^{cur}][R_{lut}^{cur}] = \max\left\{\frac{Op_{dsp}^l}{P_{fl}^l}, Lat^{pre}\right\} \quad (11)$$

where

$$\begin{aligned} Lat^{pre} &= Lat[l-1][\hat{R}_{dsp}^{pre}][\hat{R}_{lut}^{pre}] \\ \hat{R}_{dsp}^{pre} &= R_{dsp}^{cur} - \hat{R}_{dsp}^l \\ \hat{R}_{lut}^{pre} &= R_{lut}^{cur} - \hat{R}_{lut}^l \end{aligned}$$

## VII. EXPERIMENT

### A. Experiment Setup

We evaluate our framework with two datasets, the single object detection dataset in Design Automation Conference System Design Contest (DAC-SDC) [26] and CIFAR-10. For DAC-SDC, we deploy two commonly-adopted models, UltraNet [27] and SkyNet [28], with full pipeline structure, and target the bit-width selection from top-3 teams as the baseline. In order to fairly compare with previous results, we retrain and evaluate all models with the same hyper-parameter on the DAC-SDC public dataset. For CIFAR-10, we adopt a VGG-alike model (denoted as VGG-Tiny) with 6 convolution layers and one fully-connected layer, and manually design the bit-width selection as the baseline. For deployment, we target an embedded FPGA Ultra96-V2 (with 360 DSPs), and measure throughput, inference accuracy, utilization, and energy consumption for evaluation. In experiments, test frames are loaded into the DDR in advance and are then transferred to the accelerator by DMA. The throughput and energy consumption during inference are then measured following DAC-SDC

2022<sup>2</sup>. The inference accuracy is calculated with Intersection-Over-Union (IOU) for DAC-SDC and top-1 accuracy for CIFAR-10.

### B. DSP Packing Efficiency Comparison

We start with our *DSP Packing Optimizer*. As an illustrative example, Fig. 4 compares a  $3 \times 3$  kernel’s multiplication throughput lookup tables that are respectively searched by HiKonv [10] and our optimizer. As can be seen, our optimizer achieves higher multiplication throughput in 25 out of the 49 combinations. Furthermore, for  $1 \times 1$  and  $5 \times 5$  kernels, 16 and 27 cases witness throughput improvement respectively. As for the LUT overhead, we randomly sample and synthesize 30 optimized combinations. The results indicate that only 16.4 extra LUTs are required on average. These results demonstrates the effectiveness of our *DSP Packing Optimizer* in generating more efficient DSP packing configurations.

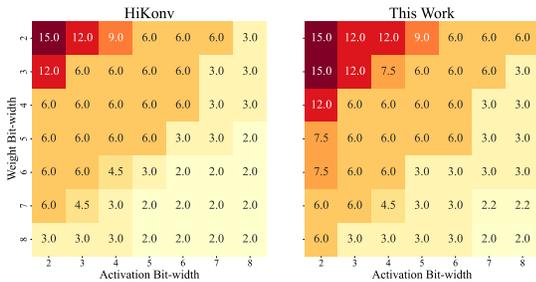


Fig. 4. DSP packing efficiency comparison for a  $3 \times 3$  convolution kernel.

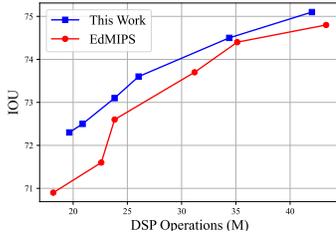


Fig. 5. DSP-aware NAS results comparison with EdMIPS [19].

### C. Bit-width Search Results

1) *NAS Comparison*: To assess our DSP-aware quantization search, we compare it with EdMIPS [19] on UltraNet. Following Eq. 6, we target DSP operations as the proxy signal to balance inference accuracy and DSP operations, while EdMIPS uses the product of activation bit-width and weight bit-width to formulate computation complexity loss. Through adjusting hyper-parameter  $\eta$ , different solutions can be obtained as the relative significance changes, as illustrated in Fig. 5. Clearly, our method produces a pareto-optimal curve with respect to both accuracy and DSP operations, which demonstrates our metrics can effectively direct the NAS to conduct DSP-aware quantization.

2) *Bit-width Selection Comparison*: Then, we compare the searched bit-width settings with the manually crafted counterparts in Fig. 6. For UltraNet, we compare with iSmart<sup>3</sup> (2nd in DAC-SDC 2021). They adopt high bit-width in the first

and last layers, and apply 4-bit weight and activation in the rest layers such that they can leverage one DSP to pack 6 multiplications. Similar quantization scheme is also applied to VGG-tiny. For SkyNet<sup>4</sup>, we quantize the weight and activation to 5 bits and 8 bits by referring to SkrSkr (1st in DAC-SDC 2021), and squeeze two multiplications into one DSP.

The experiment results demonstrate that our NAS can effectively optimize both inference accuracy and DSP operations. For accuracy, mixed-precision UltraNet, SkyNet, and VGG-Tiny respectively achieve IOU or top-1 accuracy of 74.29%, 75.44% and 91.36%, while the accuracy of handcrafted designs are 73.37%, 74.85% and 91.45%. Except for a trivial loss for VGG-Tiny, the other two models witness non-trivial accuracy improvement. As for computation complexity, 27.12%, 44.10% and 42.71% of the DSP operations are reduced.

The optimization can be explained as our NAS conducts a more reasonable bit-width allocation for each layer based on its sensitivity and computation complexity. The second and third layer of UltraNet dominate nearly 60% of the overall MACs. Hence, our NAS adopts ultra-low bit-width combinations to pack 12 multiplications onto one DSP block, and in the meanwhile, increases the bit-widths in behind layers to counteract the accuracy loss. SkyNet mainly consist of six stacked bundles of depth-wise and point-wise convolution. Due to the huge complexity dichotomy between these two types of operators, larger bit-width is preferred for depth-wise convolution. In addition, unlike SkrSkr that uniformly applies larger bit-width to activation than weights, our NAS takes the opposite strategy in several layers. For VGG-Tiny, our NAS nearly choose lower bit-width for every middle layer. This means we underestimate model’s tolerance to quantization when manually designing the bit-width settings.

### D. Deployment Results

Then, we deploy these models at different resources constrains to separately compare utilization and throughput. We firstly allocate DSPs as many as possible to deploy the manually crafted models at their highest throughput (MC-HP) that can be supported by the platform. Next, for evaluating utilization, we deliberately reduce the available DSPs to deploy our MPNNs at the nearest throughput (Mix-BP) with the corresponding baselines. Finally, we allow our framework to maximize DSP allocation (Mix-HP). Additionally, we also enable LUT-replacement (Mix-LUT) to further improve performance. All deployment results are summarized in Table I.

As can be observed, due to the reduced DSP operations, our mixed-precision models demand less parallel factors to achieve the same FPS. Compared with MC-HP, Mix-BP implementations theoretically reduce 72 (26.9%), 70 (41.7%) and 96 (36.5%) parallel factors respectively. Since DSPs can be mapped to other logic (e.g. batch normalization) as well, the final DSP savings are 100 (29.5%), 49 (21.5 %) and 100 (36.0 %). In terms of throughput, Mix-HP implementations achieve  $1.59\times$ ,  $1.71\times$  and  $1.97\times$  speedup. For UltraNet and VGG-Tiny, the highest throughput is restricted by the available DSP

<sup>2</sup>[https://github.com/jgoeders/dac\\_sdc\\_2022](https://github.com/jgoeders/dac_sdc_2022)

<sup>3</sup>[https://github.com/jgoeders/dac\\_sdc\\_2021\\_designs/tree/main/iSmart](https://github.com/jgoeders/dac_sdc_2021_designs/tree/main/iSmart)

<sup>4</sup>[https://github.com/jgoeders/dac\\_sdc\\_2021\\_designs/tree/main/SkrSkr](https://github.com/jgoeders/dac_sdc_2021_designs/tree/main/SkrSkr)

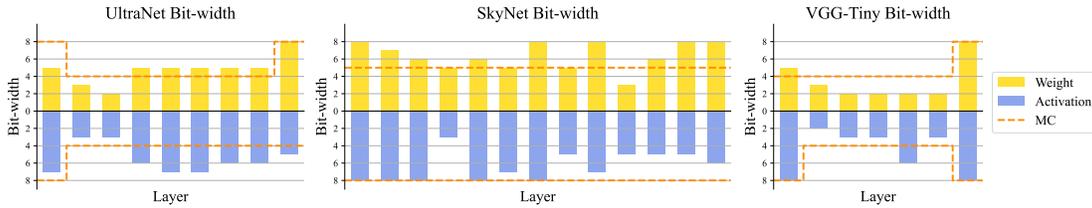


Fig. 6. Bit-width selections of our mixed-precision models and manually crafted counterparts (MC).

TABLE I  
DEPLOYMENT DETAILS OF THE MIXED-PRECISION MODELS AND MANUALLY CRAFTED COUNTERPARTS.

Backbone	Implem.	Accuracy	$Op_{dsp}$ (M)	$Pf_{dsp}$	$Pf_{lut}$	DSP	kLUT	BRAM	FPS	Power (W)
UltraNet	MC-HP	73.37%	40.78	268	0	339 (94.2%)	38.0 (53.9%)	96.0 (44.4%)	1232.10	1.38
	Mix-BP			196	0	<b>239 (66.4%)</b>	38.2 (54.2%)	108.5 (50.2%)	1234.59	1.39
	Mix-HP	74.29%	29.72	296	0	332 (92.2%)	44.6 (63.2%)	130.0 (60.2%)	1954.30	1.44
	Mix-LUT			264	128	307 (85.3%)	62.0 (87.9%)	160.5 (74.3%)	<b>2534.20</b>	1.53
SkyNet	MC-HP	74.85%	200.6	168	0	228 (63.3%)	44.2 (62.6%)	215.0 (99.5%)	193.30	2.99
	Mix-BP			98	0	<b>179 (49.7%)</b>	35.8 (50.8%)	210.5 (97.5%)	193.21	2.66
	Mix-HP	75.44%	112.13	183	0	273 (75.8%)	40.6 (57.6%)	200.5 (92.8%)	<b>330.93</b>	1.72
	Mix-LUT			119	64	209 (58.1%)	43.6 (61.8%)	200.5 (92.8%)	330.92	3.02
Vgg-Tiny	MC-HP	91.45%	25.78	263	0	278 (77.2%)	42.5 (60.2%)	213.5 (98.8%)	2011.55	1.79
	Mix-BP			167	0	<b>178 (49.4%)</b>	32.1 (45.6%)	128.0 (59.3%)	2011.29	1.43
	Mix-HP	91.36%	14.77	333	0	348 (96.7%)	46.6 (66.0%)	133.5 (61.8%)	3970.87	1.36
	Mix-LUT			307	144	340 (94.4%)	62.7 (89.0%)	158.0 (73.0%)	<b>4877.76</b>	1.44

TABLE II  
HARDWARE COMPARISON WITH PREVIOUS CO-DESIGN WORKS.

	FILM-QNN [29]	N3H-Core [30]	HAO [3]	SEUer [31]	This Work		
DSP Optimization	INT4, INT8	Bit-Parallel	INT8	HiKonv	Packing Optimizer		
Backbone	ResNet-50	ResNet-18	Searched	UltraNet	UltraNet	SkyNet	VGG-Tiny
Precision	95%W4+5%W8, A5	Mix(2-8)	W-mixed, A8	4W4A, 8FL	Mix(2-8)	Mix(2-8)	Mix(2-8)
Platform	ZCU102	XC7Z045			ZU3EG		
DSP	2092	900	360	337	307	273	340
kLUT	180.1	152.9	55.1	50.2	62.0	40.6	62.7
Frequency(MHz)	150	100	-	250	250	250	250
FPS	109.10	123.2	50.0	2084.6	2534.2	330.9	4877.8
GOPS	891.4	446.8	217.1	828.6	1007.4	263.2	1489.6
GOPS/DSP	0.426	0.50	0.60	2.46	3.28	0.96	4.38
GOPS/kLUT	4.948	2.92	3.94	16.51	16.25	6.48	23.76
Power(W)	12.9	-	5.5	1.6	1.5	1.7	1.4
Energy Efficiency (GOPS/W)	69.1	-	39.5	533.2	658.8	153.0	1099.6

resources, but our SkyNet designs suffer from lack of BRAMs which prevents us from utilizing more DSPs. After enabling LUT-replacement, our framework replaces 128, 64, and 144 DSPs respectively for the three MPNNs. This further boosts the FPS of UltraNet and VGG-Tiny to 2534.20 and 4877.76. For SkyNet, while FPS remains the same, we are surprised to find that power consumption is increased from 1.72 to 3.02 W after replacing a part of DSPs.

In Table II, we also compare our designs with prior FPGA-based accelerators that focus on quantization and implementation co-optimization. FILM-QNN [29] restricts quantization choices as either W4A5 or W8A5, and respectively applies INT4 and INT8 for packing optimization. In order to support flexible bit-widths, N3H-Core [30] implements 4-bit multiplication with DSPs, and applies bit-serial LUT-cores for others. Similarly, HAO [3] leverages INT8 for high-precision multiplications and LUTs for those under 4-bit. The champion team of DAC-SDC 22, SEUer [31], adopts the same bit-width and packing strategy as our UltraNet baseline, but they further boost the throughput by replacing the high bit-width arithmetic

with LUTs in the first and last layer, which leads to poor WNS as a result. Based on optimized DSP packing strategies, DSP-aware bit-width exploration, and fine-grained resource allocation scheme, our solutions show unparalleled throughput and arithmetic intensity compared with these designs.

## VIII. CONCLUSION

In this paper, we present a framework to co-optimize the implementation and quantization of MPNNs on FPGAs. We further optimize the state-of-the-art DSP packing algorithms to support efficient implementation of arbitrary-precision convolution arithmetic. Then, we leverage differentiable NAS for automatically crafting bit-width settings based on a comprehensive assessment of accuracy and DSP operations. Finally, with our fine-grained resources allocation scheme, pipelined accelerators are customized according to specific resource requirements. According to our experiment results, our solutions reveal superior accuracy, resource utilization, and throughput, compared with manually crafted solutions and other related designs.

## REFERENCES

- [1] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *CoRR*, vol. abs/1606.06160, 2016. [Online]. Available: <http://arxiv.org/abs/1606.06160>
- [2] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.
- [3] Z. Dong, Y. Gao, Q. Huang, J. Wawrzyniek, H. K. So, and K. Keutzer, "Hao: Hardware-aware neural architecture optimization for efficient inference," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 50–59.
- [4] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.
- [5] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: an automated tool for building high-performance dnn hardware accelerators for fpgas," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.
- [6] X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-M. Hwu, and D. Chen, "Dnnexplorer: A framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.
- [7] Xilinx, "Ultrascale architecture dsp slice," 2021. [Online]. Available: <https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp>
- [8] Intel, "Intel® arria® 10 native fixed point dsp ip core user guide," 2017. [Online]. Available: <https://www.intel.com/content/www/us/en/docs/programmable/683583/current/intel-arria-native-fixed-point-dsp-ip.html>
- [9] J. Sommer, M. A. Özkan, O. Kesocze, and J. Teich, "Dsp-packing: Squeezing low-precision arithmetic into fpga dsp blocks," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, 2022, pp. 160–166.
- [10] X. Liu, Y. Chen, P. Ganesh, J. Pan, J. Xiong, and D. Chen, "Hikonv: High throughput quantized convolution with novel bit-wise management and computation," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 140–146.
- [11] Xilinx, "Convolutional neural network with int4 optimization on xilinx devices," 2020. [Online]. Available: <https://docs.xilinx.com/v/u/en-US/wp521-4bit-optimization>
- [12] —, "Deep learning with int8 optimization on xilinx devices," 2017. [Online]. Available: <https://docs.xilinx.com/v/u/en-US/wp486-deep-learning-int8>
- [13] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Hao: Hardware-aware automated quantization with mixed precision," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8604–8612.
- [14] Y. Li, C. Hao, X. Zhang, X. Liu, Y. Chen, J. Xiong, W.-m. Hwu, and D. Chen, "Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
- [15] L. Yang and Q. Jin, "Fracbits: Mixed precision quantization via fractional bit-widths," in *AAAI Conference on Artificial Intelligence*, 2020.
- [16] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. [Online]. Available: <https://arxiv.org/pdf/1908.09791.pdf>
- [17] H. Bai, M. Cao, P. Huang, and J. Shan, "Batchquant: Quantized-for-all architecture search with robust quantizer," in *NeurIPS*, 2021. [Online]. Available: <https://arxiv.org/pdf/2105.08952.pdf>
- [18] S. Uhlich, L. Mauch, F. Cardinaux, K. Yoshizawa, J. A. Garcia, S. Tiedemann, T. Kemp, and A. Nakamura, "Mixed precision dnns: All you need is a good parametrization," in *International Conference on Learning Representations*, 2019.
- [19] Z. Cai and N. Vasconcelos, "Rethinking differentiable search for mixed-precision neural networks," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 2346–2355.
- [20] K. T. Chitty-Venkata, M. Emani, V. Vishwanath, and A. K. Somani, "Efficient design space exploration for sparse mixed precision neural architectures," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 265–276. [Online]. Available: <https://doi.org/10.1145/3502181.3531463>
- [21] W. Chen, Y. Wang, S. Yang, C. Liu, and L. Zhang, "You only search once: A fast automation framework for single-stage dnn/accelerator co-design," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1283–1286.
- [22] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10726–10734.
- [23] Y. Zhang, B. Sun, W. Jiang, Y. Ha, M. Hu, and W. Zhao, "Wsq-addernet: Efficient weight standardization based quantized addernet fpga accelerator design with high-density int8 dsp-lut co-packing optimization," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: <https://doi.org/10.1145/3508352.3549439>
- [24] A. Samajdar, T. Garg, T. Krishna, and N. Kapre, "Scaling the cascades: Interconnect-aware fpga implementation of machine learning problems," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 342–349.
- [25] R. Shi, Y. Ding, X. Wei, H. Li, H. Liu, H. K.-H. So, and C. Ding, "Ftdl: A tailored fpga-overlay for deep learning with high scalability," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
- [26] X. Xu, X. Zhang, B. Yu, X. S. Hu, C. Rowen, J. Hu, and Y. Shi, "Dac-sdc low power object detection challenge for uav applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 2, pp. 392–403, 2021.
- [27] K. Zhan, J. Guo, B. Song, W. Zhang, and Z. Bao, "Ultraneet : A fpga-based object detection for the dac-sdc 2020," 2020. [Online]. Available: [https://github.com/heheda365/ultra\\_net](https://github.com/heheda365/ultra_net)
- [28] X. Zhang, H. Lu, C. Hao, J. Li, B. Cheng, Y. Li, K. Rupnow, J. Xiong, T. Huang, H. Shi, W.-m. Hwu, and D. Chen, "SkyNet: a hardware-efficient method for object detection and tracking on embedded systems," in *Conference on Machine Learning and Systems (MLSys)*, 2020.
- [29] M. Sun, Z. Li, A. Lu, Y. Li, S.-E. Chang, X. Ma, X. Lin, and Z. Fang, "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*, ser. FPGA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 134–145. [Online]. Available: <https://doi.org/10.1145/3490422.3502364>
- [30] Y. Gong, Z. Xu, Z. He, W. Zhang, X. Tu, X. Liang, and L. Jiang, "N3h-core: Neuron-designed neural network accelerator via fpga-based heterogeneous computing cores," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 112–122. [Online]. Available: <https://doi.org/10.1145/3490422.3502367>
- [31] J. Zhang, X. Cao, Y. Zhang, G. Li, and M. Zhang, "Seuer : Dac-sdc 2022 champion," 2022. [Online]. Available: [https://github.com/AiArtisan/dac\\_sdc\\_2022\\_champion](https://github.com/AiArtisan/dac_sdc_2022_champion)