A Reconfigurable Depth-Wise Convolution Module for Heterogeneously Quantized DNNs

A Reconfigurable Depth-Wise Convolution Module for Heterogeneously Quantized DNNs

Luca Urbinati and Mario R. Casu
Department of Electronics and Telecommunications
Politecnico di Torino, 10129 Turin, Italy
{luca.urbinati, mario.casu}@polito.it

***Abstract*—In Deep Neural Networks (DNN), the depth-wise separable convolution has often replaced the standard 2D convolution having much fewer parameters and operations. Another common technique to squeeze DNNs is heterogeneous quantization, which uses a different bitwidth for each layer. In this context we propose for the first time a novel Reconfigurable Depth-wise convolution Module (RDM), which uses multipliers that can be reconfigured to support 1, 2 or 4 operations at the same time at increasingly lower precision of the operands. We leveraged High Level Synthesis to produce five RDM variants with different channels parallelism to cover a wide range of DNNs. The comparisons with a non-configurable Standard Depth-wise convolution module (SDM) on a CMOS FDSOI 28-nm technology show a significant latency reduction for a given silicon area for the low-precision configurations.**

***Index Terms*—Deep Neural Networks, Reconfigurable Hardware, Mixed-Precision Quantization, Depth-wise Convolution.**

I. Introduction

To reduce computing and memory requirements of Deep Neural Networks (DNNs) on edge devices, an effective technique is mixed-precision quantization, which aims to quantize DNN layers with different precision. In fact, [1] showed that input and weight statistics differ greatly among layers and [2] proved that each layer requires a different precision. Another solution to compress DNNs is the Depth-wise Separable Convolution (DSC), introduced in [3], become popular with MobileNet-V1 [4] and found in many lightweight models [5]–[9]. DSC reduces the number of parameters and operations by replacing the standard 2D convolution with two subsequent blocks, depth-wise and point-wise, at a small accuracy penalty.

In this context, we propose for the first time a novel Reconfigurable Depth-wise convolution Module (RDM) that uses multipliers with a Sum Together (ST) mode [10] in its Multiply-and-Accumulate (MAC) units. This mode allows packing N low-precision (activations, weights) pairs as the multiplier inputs, and computing their dot-products at reduced precision in one shot. Therefore, this saves the additional $N-1$ MAC operations that a non-configurable MAC unit would require. In particular, our RDM supports packing N (activation, weight) pairs at precision between 1 and $16/N$ bits, where N = 1, 2 or 4, resulting in 1×(1-16 bits), 2×(1-8 bits) or 4×(1-4 bits). We define these supported configurations as 16x, 8x and 4x, respectively.

We leveraged High Level Synthesis (HLS) to quickly produce five RDM variants that can compute 1, 2, 4, 8, or 16 channels in parallel. We performed a Design-Space Exploration (DSE) to find the Pareto frontier in the Latency vs. Area and Energy vs. Area spaces. We compared our designs with a non-configurable Standard Depth-wise convolution module (SDM) explained in Sec. IV in more detail. Results obtained on an FDSOI 28-nm technology show a significant latency reduction for a given silicon area at a small energy cost.

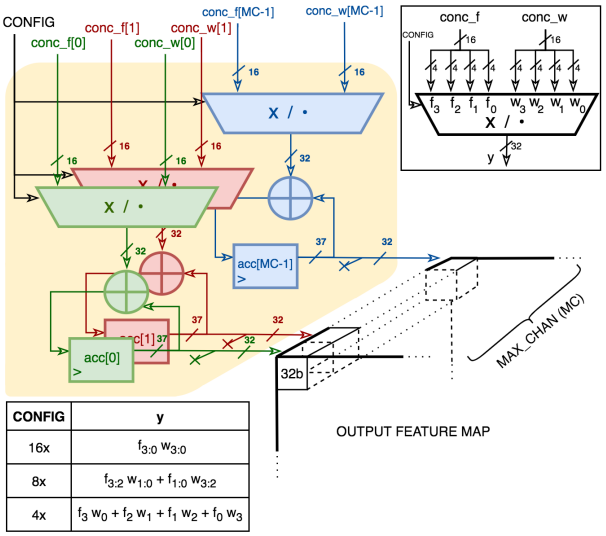
II. Related Work

In the first era of DNN accelerators the inference computation was mainly executed by constant-precision operators: DaDianNao [11], EIE [12], and Eyeriss v1 [13] use 16-bit fixed-point operators, while Minerva [14] tries to quantize with lower precision. Today, with advanced strategies we can reduce the bitwidth of both weights and activations as low as 2-bit [15] preserving accuracy. Hence, a new generation of accelerators started to exploit reduced data types. For example, Bit Fusion [16] composes and decomposes 2-bit multipliers to support 8-bit/2-bit, 4-bit/4-bit, 2-bit/8-bit and 8-bit/8-bit configurations for inputs/weights. Thinker [17] adopts a bit-width adaptive computing unit, which can be configured to execute two 8 × 16-bit multiplications in parallel or one 16 × 16-bit multiplication. Zhou et al. [18] deploys two types of Processing Elements (PEs) on a FPGA accelerator to separately process full (16-bit) and low (8-bit) precision operations. DNPU [19] uses look-up table-based reconfigurable multipliers that support 4-/8-/16-bit multiplications, while UNPU [20] uses serial multipliers to implement lookup table-based PEs to enable precisions from 1 to 16 bits. BISMO [21] uses a bit-serial dot product unit for FPGAs that can be utilized for a range of different precisions. ENVISION [22] is an ASIC that proposes a MAC unit with multipliers that support the same precisions of our design , but work in Sum Separate (SS) mode [10].

In the area of microcontrollers, XPulpNN [23] integrates a multiple-precision dot-product unit in RISC-V, featuring SIMD vectors of 16- down to 2-bit precision elements. Since it is followed by an adder tree that sums up the partial products, it computes in SS mode.

Commercial products already incorporate mixed-precision accelerators, like Apple A12 Bionic chip [24], NVIDIA Turing GPU [25], and Socionext Inc. NPU [26].

When it comes to DSC, there are just few examples of accelerators that support it, such as: [27] and [21] for FPGA; [28]–[30] for ASIC, but only [21] supports multi-precision DSC. Finally, if we did not overlook some previous work, none of these leverage the flexibility of HLS for a quick DSE.



CONFIG	y
16x	$f_{3:0}\ w_{3:0}$
8x	$f_{3:2}\ w_{1:0} + f_{1:0}\ w_{3:2}$
4x	$f_3\ w_0 + f_2\ w_1 + f_1\ w_2 + f_0\ w_3$

Fig. 1: RDM MAC Unit array with reconfigurable multipliers

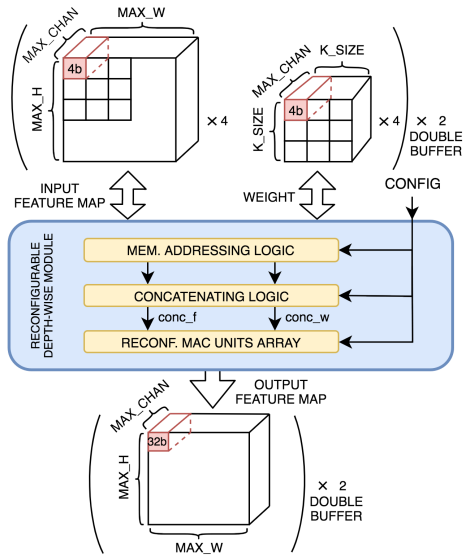


Fig. 2: Overview of the RDM.

III. Hardware Architecture

The RDM contains the MAC Units array shown in Fig. 1. MAX_CHAN (MC) corresponds to the number of ST reconfigurable multipliers instantiated in the MAC Units array and this number corresponds to the number of channels the RDM can process in parallel. Each unit takes the two 16-bit operands and unpacks them to feed the ST multiplier according to the configuration represented by the CONFIG signal. The table inside Fig. 1 reports the three different operations done by an ST reconfigurable multiplier. An accumulator takes the ST multiplier output and accumulates it until the number of iterations for a convolution between a kernel and a feature map receptive field is completed. After that, the result is cast to a 32-bit number and stored in an output buffer for subsequent computations.

An overview of the RDM architecture can be seen in Fig. 2. It includes a memory architecture with double buffers, an addressing logic, a packing logic and the reconfigurable MAC Units array. The memory architecture for the input feature-map and weight tensors consists of four 4-bit memories, each with shape (MAX_W×MAX_H×MAX_CHAN) and (MAX_K_SIZE×MAX_K_SIZE×MAX_CHAN), respectively. Later, we refer to the feature map 4-bit blocks as A_F, B_F, C_F and D_F, and to the weight 4-bit blocks as A_W, B_W, C_W and D_W. Instead the output memory is a single 32-bit memory with shape (MAX_W×MAX_H×MAX_CHAN).

Since the size of DSC feature map tensors and weight tensors can exceed the size of the memory of the RDM in one or multiple dimensions, it is necessary to iterate over several tiles to complete a depth-wise convolution. The number of iterations depends on the value of its design-time configuration parameters: MAX_W (=MAX_H) and MAX_K_SIZE. To derive the values of MAX_W and MAX_K_SIZE, we analyzed not only the most cited DNNs for classification and object detection, but also their presence in some public Model Zoos for edge devices, such as Google Coral, TensorFlow Hub, Intel, Xilinx VitisAI and Nvidia. We finally selected EfficientNet-B0 [7], MobileNetV1 [4], MobileNetV2 [5] and the SSD and SSD-Lite versions of those two last networks. From our analysis we found that setting MAX_W = 22 was a reasonable trade-off to keep the area overhead of the memories around two times the area of the logic and to limit the number of RDM iterations. Regarding MAX_K_SIZE, since the most common kernel sizes were 3x3 and 5x5, we decided to use MAX_K_SIZE = 5. As for the MAX_CHAN design-time parameter, we performed the DSE described in Sec. IV.

Since our accelerator will be part of an SoC for edge applications, an embedded processor or a DMA engine will fill those memories as follows. For simplicity, consider only one channel of the input feature-map tensor. In the 16x case, the values are extended to 16-bit (if needed), then split into *four* 4-bit chunks, and stored in order from most to least significant into A_F-D_F. The procedure is repeated for all the channels and it holds for the weight buffer as well. In the 8x case, the values are extended to 8-bit (if needed), split into *two* 4-bit chunks, and stored in C_F-D_F. Finally, in the 4x case each element is extended to 4-bit and stored in D_F.

Feeding all the MAC Units array in parallel requires a particular memory addressing and concatenating logic. Imagine a dummy feature-map tensor with shape 5×5×MAX_CHAN and a weight tensor with shape 3×3×MAX_CHAN. This means that the weight filter creates a 3×3 receptive field on the feature map tensor, as shown in Fig. 3.

1) 16x configuration: The RDM reads MAX_CHAN 4-bit elements along the channel axis from the same position in the receptive field in all the four input feature memories A_F, B_F, C_F and D_F, as shown in Fig. 3(a). This is possible because the four memories are interleaved with factor MAX_CHAN through an HLS directive. The four MAX_CHAN long arrays are then concatenated element-wise to form a single 16-bit array in the order expressed by the following equation:

$$conc\_f[c] = A_F[i][c] \& B_F[i][c] \& C_F[i][c] \& D_F[i][c]$$

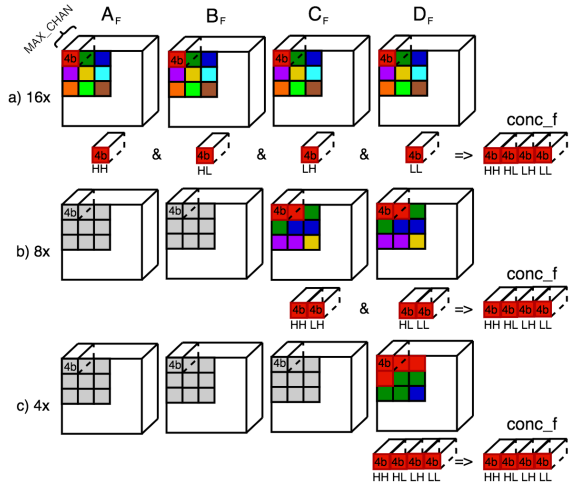


Fig. 3: Memory addressing and concatenation of the input feature-map tensor.

where c is one of the channels in $\{0, \ldots, \text{MAX\_CHAN} - 1\}$, i is the position index inside the receptive field in $\{0, \ldots, K_\square - 1\}$, where $K_\square = \text{K\_SIZE} \times \text{K\_SIZE}$, and & is the concatenation operator. The same procedure is applied to the weights A_W, B_W, C_W and D_W. The result is a MAX_CHAN long 16-bit array, called $conc\_w[c]$.

2) 8x configuration: The RDM reads MAX_CHAN 4-bit elements along the channel axis which correspond to *two* consecutive positions in the receptive field of the *two* memories C_F and D_F, as shown in Fig. 3(b) in red. Then the four, MAX_CHAN long arrays are concatenated element-wise to form a single 16-bit array, but the order of concatenation is different than the 16x case: two 8-bit operands must be placed side by side to form a single 16-bit operand to perform the dot-product as explained in Fig. 1. For the weights, the steps are the same, but they are applied to C_W and D_W, and the concatenation order is reversed:

$$conc\_f[c] = C_F[i][c] \& D_F[i][c] \& C_F[i{+}1][c] \& D_F[i{+}1][c]$$
$$conc\_w[c] = C_W[i{+}1][c] \& D_W[i{+}1][c] \& C_W[i][c] \& D_W[i][c]$$

3) 4x configuration: For this case (Fig. 3(c)), we have:

$$conc\_f[c] = D_F[i][c] \& D_F[i{+}1][c] \& D_F[i{+}2][c] \& D_F[i{+}3][c]$$
$$conc\_w[c] = D_W[i{+}3][c] \& D_W[i{+}2][c] \& D_W[i{+}1][c] \& D_W[i][c]$$

In general, the number of MAC cycles required to get MAX_CHAN output *pixels* is $\lceil K_\square / N \rceil$, where N = 1, 2 or 4 for the configuration 16x, 8x or 4x, respectively; while the theoretical speedup achievable by the RDM is $K_\square / \lceil K_\square / N \rceil$. The actual speedup is less than the theoretical value due to control logic overhead and ranges from 1.3 for the 8x configuration to 1.6 for the 4x one.

IV. Experimental Results

Through our analysis of selected lightweight DNNs (Sec. III), we found that 16 is the greatest common divisor of the channels of any depth-wise convolution layer. Thus, we made a DSE using Catapult HLS by sweeping MAX_CHAN from 1 to 16 in power of 2 values, and the operating clock frequency f_{clk} from 100 to 1000 MHz in ten steps. We synthesized the RTL netlists generated by Catapult HLS with Synopys Design Compiler (DC). We compared our RDM with a non-reconfigurable accelerator (SDM) based on a standard 16-bit multiplier that uses sign extension whenever required for low-precision operands. The SDM does not perform dot-products at reduced precisions inside the MAC Units array, which results in simpler Memory Addressing, Concatenating logic and MAC Units array, but a higher number of MAC operations for lower precisions. We analyzed the performance of RDM and SDM over two significantly different depth-wise layers of MobileNetV1, operating on (112×112×32) and (7×7×1024) feature map tensors, respectively, and both with (3×3) filters[1]. We realized that the results follow the same trend, hence we report only those obtained on the last layer.

Fig. 4a, 4c and 4e report Latency vs. Area with the modules configured to work in 16x, 8x and 4x mode, respectively; in the same way Fig. 4b, 4d and 4f show Energy vs. Area. The results take into account the iterations required because the tensor sizes exceed the maximum size that DRM and SDM support. The latency is the total number of clock cycles multiplied by the clock period; the energy is the power estimated by DC multiplied by the latency to which we add the memory energy that we estimated using the same model of [31].

From the plots in Fig. 4 we can conclude:

- As expected, the standard solutions outperform the reconfigurable ones in the 16x case because of their lower area overhead of the memory addressing and concatenating logic and of the MAC Units array (Fig. 4a). In fact, the SDM has a simpler memory addressing, does not reorder the operands, and uses non-reconfigurable multipliers.
- Reconfigurable Pareto solutions in configuration 8x and 4x have indeed a lower latency for a given area than standard ones with the same MAX_CHAN.
- The lower latency is paid with a slightly higher energy because the RDM Pareto points in the Latency vs Area space (see the *reddish* colors in Fig. 4c vs. 4d and in Fig. 4e vs. 4f) have higher clock frequencies than the SDM ones. Notice, however, that the scale of the y-axis is linear for energy and logarithmic for latency. Thus, the compromise is acceptable.
- For latency in 8x and 4x, 30% and 50% of the reconfigurable points are on the Pareto frontier, respectively.

V. Conclusion

We presented a Reconfigurable Depth-Wise Convolution Module for Heterogeneously Quantized DNNs and compared it with a standard non-reconfigurable module. The results of the design-space exploration show a trade-off between latency and area in favor of the reconfigurable solutions especially when low-precision quantization is used. This latency advantage is paid with a slight energy penalty. We plan to complete our accelerator by adding a point-wise convolution module that will use the same reconfigurable MAC units.

[1] All MobileNetV1 depthwise layers use the same kernel size.

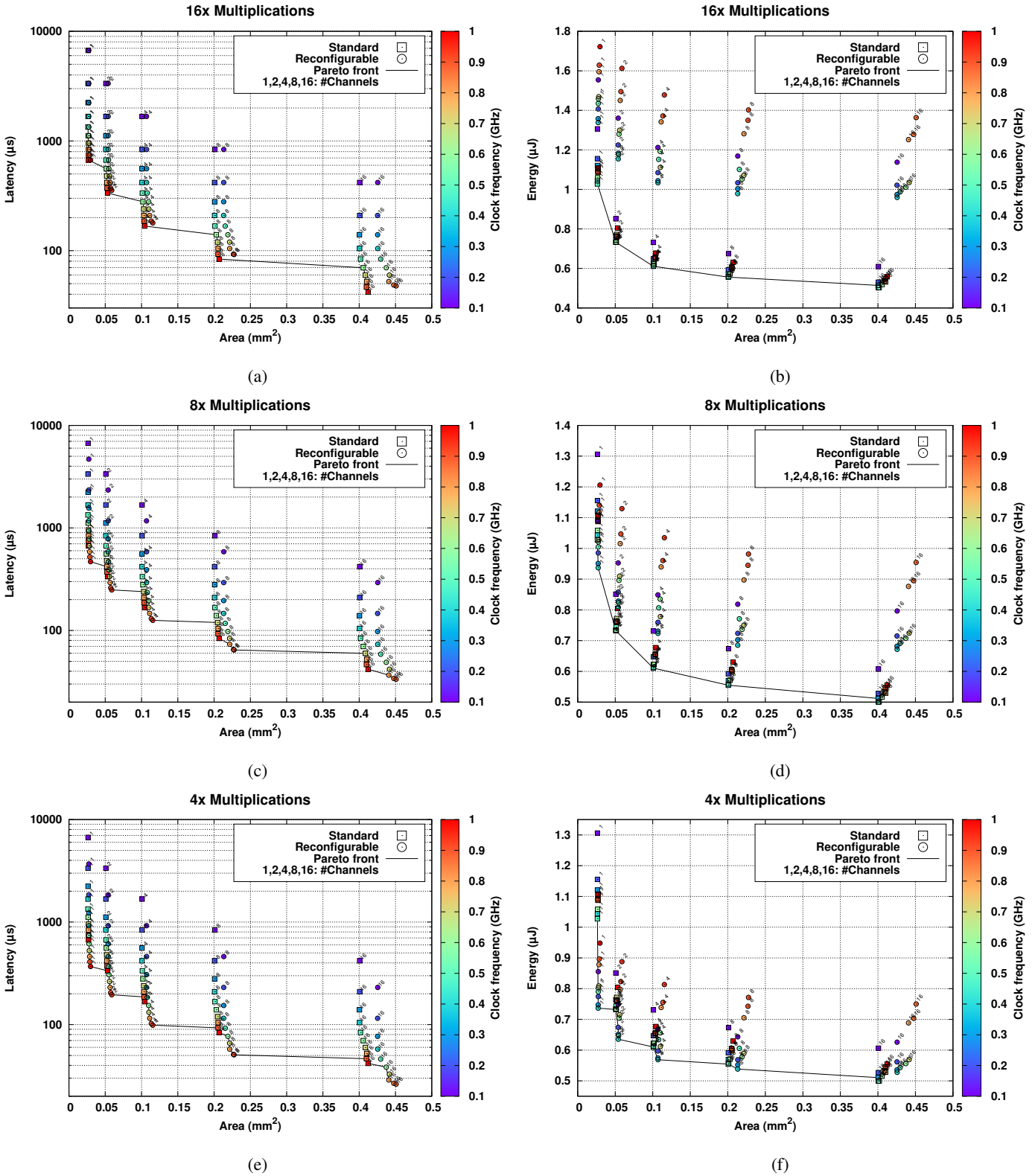


Fig. 4: Latency vs. Area and Energy vs. Area DSEs for the last layer of MobileNetV1.

REFERENCES

[1] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016, pp. 1–8.

[2] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 1131–1135.

[3] V. Vanhoucke, "Learning visual representations at scale," *ICLR invited talk*, 2014.

[4] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[6] A. Howard *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[7] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 6105–6114.

[8] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[9] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[10] L. Mei *et al.*, "Sub-word parallel precision-scalable mac engines for efficient embedded dnn inference," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2019, pp. 6–10.

[11] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609–622.

[12] S. Han *et al.*, "Eie: Efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 243–254.

[13] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379.

[14] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 267–278.

[15] X. Sun *et al.*, "Ultra-low precision 4-bit training of deep neural networks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1796–1807.

[16] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 764–775.

[17] S. Yin *et al.*, "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 968–982, 2018.

[18] X. Zhou, L. Zhang, C. Guo, X. Yin, and C. Zhuo, "A convolutional neural network accelerator architecture with fine-granular mixed precision configurability," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[19] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 240–241.

[20] J. Lee *et al.*, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2019.

[21] Y. Umuroglu, L. Rasnayake, and M. Själander, "Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 307–3077.

[22] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 246–247.

[23] A. Garofalo, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "Xpulpnn: Enabling energy efficient and flexible inference of quantized neural networks on risc-v based iot end nodes," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1489–1505, 2021.

[24] "Apple describes 7nm a12 bionic chips," *EENews*, 2018. [Online]. Available: https://www.eenewsanalog.com/news/apple-describes-7nm-a12-bionic-chip

[25] Nvidia, "Nvidia turing gpu architecture," 2018. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[26] T. Isono *et al.*, "A 12.1 tops/w mixed-precision quantized deep convolutional neural network accelerator for low power on edge / endpoint device," in *2020 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2020, pp. 1–4.

[27] B. Li *et al.*, "Dynamic dataflow scheduling and computation mapping techniques for efficient depthwise separable convolution acceleration," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 8, pp. 3279–3292, 2021.

[28] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[29] N. P. Jouppi *et al.*, "A domain-specific supercomputer for training deep neural networks," *Commun. ACM*, vol. 63, no. 7, p. 67–78, Jun. 2020.

[30] K.-W. Chang and T.-S. Chang, "Vwa: Hardware efficient vectorwise accelerator for convolutional neural network," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 145–154, 2020.

[31] G. Santoro, M. R. Casu, V. Peluso, A. Calimera, and M. Alioto, "Design-space exploration of pareto-optimal architectures for deep learning with dvfs," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.