More AddNet: A deeper insight into DNNs using FPGA-optimized multipliers

Martin Hardieck*[1], Tobias Habermann†[1], Fabian Wagner*, Michael Mecik†, Martin Kumm†, Peter Zipf*
*University of Kassel, Germany, {martin.hardieck, fabian.wagner, zipf}@uni-kassel.de,
†Fulda University of Applied Science, Fulda, Germany, {tobias.habermann, michael.mecik, martin.kumm}@cs.hs-fulda.de

***Abstract*—We present a training tool flow for deep neural networks (DNN) optimized for a hardware-efficient FPGA-implementation based on reconfigurable constant-coefficient multipliers (RCCMs). RCCMs replace the costly generic multipliers by shift-and-add operations. In previous work, it was shown that RCCMs offer a better alternative for saving FPGA area than utilizing low-precision arithmetic. This work proposes an improved tool flow that enables layer-wise weight quantization, a larger search space by additional RCCM coefficient sets and an optimized retraining. This leads to an improved accuracy compared to the previous method. In addition, hardware requirements are lower as only 1 to 3 adders per multiplication are used. This reduces the overall complexity and the required memory bandwidth simultaneously. We evaluate our tool flow using multiple networks (ResNets) on the ImageNet data set.**

***Index Terms*—FPGA, Neural Network Accelerator, Arithmetic, Neural Network Training, Tool flow, Fixed-Point, Low-Precision**

I. Introduction

Deep Neural Networks (DNNs) are increasingly important for many application areas from embedded systems [1] to data centers [2]. Thus, their area and energy efficient implementation is becoming a major concern [3]. In this context, field-programmable gate array (FPGA) implementations have shown the capability to outperform CPU or GPU based inference [4], [5]. FPGAs offer the ability to adapt arithmetic implementations to characteristics of a specific DNN, but this regularly requires a parameter quantization to reduce area demands. Thus, they increase efficiency at the price of a largely increased design effort [4].

AddNet [6] investigates the application of reconfigurable constant-coefficient multipliers (RCCMs) to replace generic, low-precision multiplications in FPGA-based DNN implementations. RCCMs are a special type of time-multiplexed Multiple Constant Multipliers (MCM [7]), which allow for runtime switching between weights in an efficient way. As they only use adders, subtractors, bit shifts and multiplexers, they are less costly than generic multiplications, even for low-precision arithmetic [6]. Additionally, due to the specific encoding scheme for possible coefficient values, a reduction in weights storage can be achieved.

A specific training scheme is required to adapt RCCM selection and the DNN's coefficient values distribution by mapping RCCM coefficient representations to the DNN's weights distribution, thus minimizing quantization loss. This sets the ground for the principal approach to reduce the mismatch between DNN computations and FPGA device architecture by replacing low-precision arithmetic by more suitable RCCMs. The idea in AddNet is to use a two-step training procedure, starting with an already trained network represented by floating-point values as the input. In the first step, a fixed-point training is used to quantize the input network. In the second step, the quantized version is retrained to adapt its weights to the specific values provided by RCCMs. The RCCMs with their resulting values are selected based on the weight distribution of the quantization-aware training (QAT). Furthermore, all AddNet RCCMs are based on a symmetric value distribution around zero and also have a fixed structure.

Previous work [6] already demonstrates that this kind of device architecture adaptation leads to better implementations in terms of area savings. In this paper, we improve this approach by considering the absolute quantization error when adapting the RCCMs and, in contrast to AddNet, quantization and selection of RCCM coefficients are applied on a per-layer basis. Also, a larger variety of RCCMs is provided and the training, as well as the accuracy evaluation, are improved. The main contributions of this paper are:

- the extension of the reachable search space by the introduction of additional connection structures for RCCMs in combination with a larger set of used operations (Sec. III),
- an adapted training for improved layer-wise fixed-point range scaling and a focus on only the significant weights for word size considerations (Sec. II),
- an improved and automated RCCM selection for the quantization-aware training cycle (Sec. IV),
- a modified accuracy metric to address quantization related ambiguities (Sec. II-B).

II. Quantization Aware Training with MQUAT

Several frameworks exist that allow QAT, e.g., [8]–[10]. Even small variations in the training process can lead to very different results concerning actual weights or overall accuracy of the quantized network. They can even lead to instabilities of the results, including complete accuracy deterioration. As our approach capitalizes on quantization using specific weight distributions while retaining accuracy, these observations are of paramount interest here.

[1]These authors contributed equally to this work.

We refined the open-source *Modular Quantization Aware Training* (MQUAT) framework[1] for this work as described below. MQUAT is an extension of the TensorFlow 2 framework and uses an interface structure similar to Keras. MQUAT allows to optionally quantize activations, weight-values, bias-values, summations in the matrix multiplications and layer outputs separately. A specified quantization only applies during inference; the weights are still saved as floating-point values and can therefore slowly drift during backpropagation, following the concept of a straight-through estimator (STE) [11]. However, we suppress gradients that would result in values outside the specified quantization range.

The training tool flow used here performs a layer-based fixed-point quantization followed by a matching of RCCM coefficient sets to the weight value distributions of the quantized network layers. This approach results in a tight architecture adaptation of the arithmetic.

A. Adaptation of the Quantization to the Value Distribution

Quantization of the floating-point weights is done by a stepwise reduction of the word size available for the values accompanied by retraining the network, thus slowly adapting the quantization values to the original values. This is necessary as directly using the target word size usually leads to bad accuracy results. By clipping outliers, only the most relevant range is considered for quantization [12], [13]. This is implemented in MQUAT by ignoring all outliers that are larger than a user-defined factor of the standard deviation, thus reducing the value range of the input floating-point numbers and enabling a better fit of the fixed-point representations.

To match the fixed-point range to the distribution of a clipped floating-point value set W, a layer-wise scaling is performed, based on the value-distributions of the occurring weights W_W, biases W_B, and activations W_A. While the values of W_W and W_B are embedded in the trained network, we determine the activations W_A using data from the first calculated inferences. These values are used for scaling the weights, biases, and activations individually:

We calculate the position of the binary point such that the resulting fixed-point value range is minimal and just includes the largest and smallest occurring floating-point values.

Then, we try to move this binary point position one bit to the left, i.e., moving one additional digit from the integer part to the fraction part. This means that the largest value in W now is outside the integer range. We only keep this new position if the resulting distance between the largest and smallest fixed-point and floating-point values is not too large. In our experiments, an empirically selected maximum distance of 5% worked reasonably well.

This addresses the cases where the scaling factor representing the binary point position shift was almost one position less; the extended fraction range then delivers a more accurate resolution to match the weight values in W.

Due to the asymmetry of the two's complement representation, the consideration must be carried out separately for the positive and negative fixed-point bounds and the largest resulting range extension must be used.

[1]MQUAT: www.uni-kassel.de/go/mquat

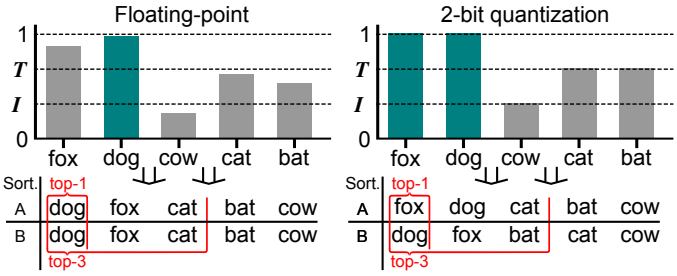

Fig. 1. Example for quantizing a prediction vector of a classifier: floating-point values (left), two bit quantization (right). The respective maximum evaluation is highlighted in green. Sorting result for top-k acc. evaluation from two different (A and B) sorting options.

B. Accuracy Metric for Low Word Sizes

The general approach for a classifier to determine a top-k metric from a prediction vector is to take the numerically highest k class predictions and check if the correct class is present as shown in Fig. 1 (left). As shown on the right side of Fig. 1, quantization to low word sizes notoriously results in tie situations, which must be resolved.

We solved this by defining $N_>$ as the number of prediction values greater than the prediction value for the correct class, and $N_\geq$ as the number of prediction values greater than or equal to the prediction value for the correct class. For example, in Fig. 1, $N_> = 2$ and $N_\geq = 4$, if *cat* was the correct class. If $N_>$ is larger or equal to k, the top-k accuracy metric is 0, because neither the correct class nor other classes with the same prediction value are in the top-k. Otherwise, we have the tie case in which the conventional top-1 accuracy depends on the sorting of the outputs. Hence, we define the metric to be equal to $\max(k/N_\geq, 1)$ to be independent of the actual sorting. For example, the top-1 accuracy in Fig. 1 leads to $\frac{1}{2}$ instead of a random 0 or 1 result. With this, ambiguous cases are penalized during training.

III. Search Space and RCCM Composition

In AddNet [6], the RCCM connection structures (CS) as shown in Fig. 2 were filled with pre-defined base cell (BC) combinations. We extend the search space by a larger operation set for the base cells and integrate the BC selection for each placeholder of the RCCM structure into the training process.

A. Low-level Implementation

The RCCM BCs used in this work are shown in Fig. 3. The data path structures are designed to use the same FPGA resources as one standard ripple carry adder. For each of the cases A, B, and C all data path options are shown. Using the select lines s, the data path for the adder can be controlled via the multiplexers. The combined select lines of the BCs become the select lines of the RCCM. The mapping from s to a specific data path configuration for a specific BC is defined by a selection function $\sigma(s)$, which is shown as a function

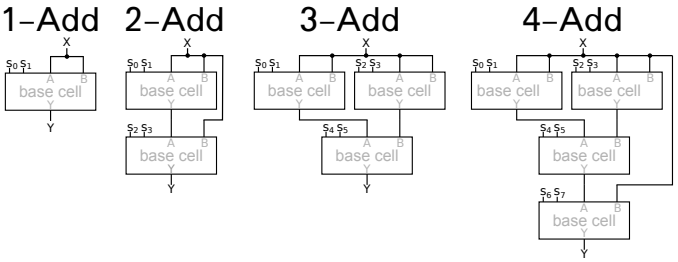

Fig. 2. RCCM connection structures 1-Add up to 4-Add. Each base cell is shown as one box with the data inputs A and B and the select lines s_i. Input value of each RCCM is X and the output of each RCCM is Y

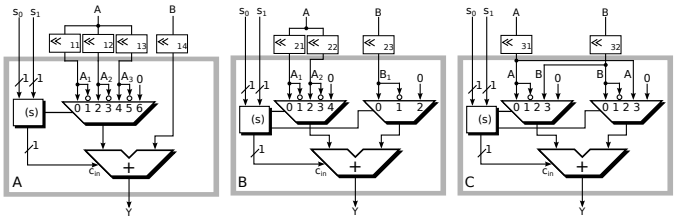

Fig. 3. Low-level base cell implementations: A, B from [6], C is a new structure. The gray boxes can be implemented in LUT resources, the parts outside the boxes in the routing network.

block in each BC. However, for an implementation, each BC uses two bits of s and $\sigma(s)$ is chosen and fixed to just four combinations of the multiplexers. During runtime, it is possible to switch between these four cases using s [6].

Each BC implementation is geared to utilize low-level features of the target FPGA architecture to the greatest possible extent. However, while cells A and B were designed to address the Xilinx Virtex [14] architecture, cell B can also be implemented on Intel FPGAs. BC C is introduced to address the specific device architecture of the Xilinx Versal [15] slices and shows that an RCCM implementation is possible. The considered CS are presented in Fig. 2 and are independent of any target device. Here, 1-Add is newly introduced as it was not possible to handle 1-Add using the selection procedure of [6]. The 4-ADD has been removed as the 3-ADD already reaches a sufficiently good accuracy with the proposed approach.

To support all possible RCCM configurations, we designed a generic open-source code generator based on FloPoCo [16], while AddNet code generation only enabled the specific RCCM structures used there.

B. RCCM Search Space

To specify a set of coefficients, the placeholders of an RCCM CS are individually filled with BCs, which in turn are specified by their type, shifts, and a chosen $\sigma(s)$. The RCCM search space consists of literally trillions of different possibilities. So, restrictions are necessary to create a usable subset for RCCM selection.

These restrictions split into three categories: available operation combinations for $\sigma(s)$, usable BC types at specific positions in the CS, and the allowed maximal shift S_{max} for the shifts φ_{xy} of each BC.

TABLE I
OVERVIEW OF RCCM SEARCH SPACE WITH USED RESTRICTIONS FOR 1-ADD, 2-ADD AND 3-ADD, OCCURRING CONFIG CASES AND RESULTING UNIQUE COEFFICIENT SET COUNTS

Type	$\sigma(s)$	Pos. 1	Pos. 2	Pos. 3	S_{max}	cases	unique
1-Add	all	A/B	-	-	8	275,130	97,727
2-Add	reduced	A/B	A/B	-	6	1,270,129	832,585
3-Add	reduced	A	A	B	4	2,000,000	1,163,742

The used restrictions for all RCCM Types are shown in Table I together with the count of occurring configurations (cases) and the resulting unique coefficient sets to choose from. For $\sigma(s)$, a reduced set results in less options to choose from which depends on the used BC. For type A, a reduced σ results in four sets. All of them contain $\{$A1+B1; A2+B1; A3+B1$\}$ and differ in $\{$B1$-$A1$\}\{$B1$-$A1$\}\{$B1$-$A1$\}\{$B1$\}$. For type B, a reduced σ results in one set: $\{$A1$-$B1; B1$-$A1; A2$-$B1; B1$-$A2$\}$

Which BC is allowed to be used at which position in the CS is defined in columns three to five of Table I. As mentioned before, not all BC types can be implemented on each FPGA architecture and therefore the target platforms influence the possible search space. In this paper, we focus on Virtex FPGAs and as type C is inefficient for Virtex, it is not used here.

The extended search space enables more possible coefficient sets, which are no longer restricted to symmetric distributions.

IV. RCCM-AWARE TRAINING

To use an RCCM for a specific computation, its coefficient set must be selected out of the numerous cases given in Table I. The idea behind this selection is similar to that for scaling regularly quantized fixed-point numbers: Based on the clipped value set of the floating-point weight values W, an RCCM providing a suitable weights distribution must be found. As the implemented weights of an RCCM depend on the selected coefficient set, the best-fitting coefficient set must be determined. As usually, batch normalization [17] computation is used, which cannot be directly implemented by RCCMs. We already fuse batch normalization effects before quantization into the weights and, subsequently, RCCM selection.

The search space can be seen as a set $\mathcal{K}$ containing all representable coefficient sets for a given RCCM architecture. The goal is to find the coefficient set $K \in \mathcal{K}$ which results in the smallest possible quantization error when used to reassemble W. To compare a coefficient set K with W, the sum of absolute quantization errors is given as

$$\delta_K = \sum_{w \in W} \min_{k \in K}(|w - k|) \ . \tag{1}$$

Similar to Sec. II-A, we compute a layer-wise bit shift for scaling the coefficient sets in $\mathcal{K}$ to W. In contrast to the fixed-point scaling, for RCCMs we only take the largest absolute values $k_{\text{max}} = \max_{k \in K}(|k|)$ and $w_{\text{max}} = \max_{w \in W}(|w|)$ of both sets

TABLE II
STANDARD DEVIATION FACTORS USED FOR CLIPPING IN EXPERIMENTS. HERE, W, B, A STANDS FOR QUANTIZATION OF WEIGHTS, BIASES, AND ACTIVATIONS, RESPECTIVELY

	Fixed-point					RCCM		
ResNet-50	6-Bit	7-Bit	8-Bit	9-Bit	10-Bit	1-Add	2-Add	3-Add
w,b	3.25	4.0	4.0	7.0	10.0	2.5	5.5	8.0
w,b,a	3.25	4.0	4.0	7.0	10.0	2.5	5.5	8.0
ResNet-18	6-Bit	7-Bit	8-Bit	9-Bit	10-Bit	1-Add	2-Add	3-Add
w,b	3.0	7.0	7.0	7.0	10.0	2.5	5.5	8.0
w,b,a	3.25	3.25	4.0	6.0	9.0	2.5	5.5	8.0

into account for determining the scaling factor $w_{\max}/k_{\max}$ and, from there, the according bit shift as $r = \lfloor \log_2(w_{\max}/k_{\max}) \rfloor$. By using bit shifts r, each $K \in \mathcal{K}$ can be correctly scaled to $2^r K$, denoting the element-wise multiplication of K with 2^r.

Typically, the scaling factor will not be a power of 2 and in that case, in addition to evaluating metric $\delta_{2^r K}$ according to (1), we also evaluate $\delta_{2^{r+1} K}$. Finally, we use the shift value (r or $r+1$) resulting in the lower δ_K for RCCM scaling. Then, the network is retrained with the selected and scaled RCCMs.

V. EXPERIMENTAL RESULTS

We performed training experiments to evaluate our approach. We do not present new synthesis results, as the generated hardware differs from [6] only by the omission of scaling multipliers which are not needed anymore.

We use the pretrained full-precision weights and augmentation settings from [18]. We retrained ResNet models [19] on the ImageNet 2012 dataset [20] with our QAT method. All experiments use the standard SGD optimizer (lr=10^{-5}, momentum=0.94, nesterov=True) and categorical-cross-entropy loss. The outlier clipping settings for all experiments are shown in Table II. All clipping settings are determined by validating the accuracy of the PTQ of each model.

To evaluate the influence of quantized activations, we present the resulting accuracy for a fixed-point only training in Table III and the results for quantizing the biases and the activations with fixed-point and the weights with a 2-Add RCCM are shown in Table IV. For this approach, an activation word size between 6 and 8 bits is sufficient. Larger word sizes have not shown any improvement.

Table V shows our main results compared with the state of the art. Here, only the weights and biases are quantized similar to [6]. Note that the floating-point accuracy has advanced a lot since [6]. Hence, to enable a fair comparison despite the differences in the baseline accuracy, we compare the relative top-1 accuracy (rel.) to the corresponding baseline and mark the superior values bold. It can be seen that the proposed method results in a significantly lower accuracy drop. This allows the usage of 3-Add instead of 4-Add as before, which saves hardware resources and reduces the memory bandwidth by 25%. It should also be noticed that it is more complex to perform non-destructive retraining of networks with high accuracy compared to low accuracy.

TABLE III
ACCURACY RESULTS (%) FOR MODELS WITH DIFFERENT FIXED-POINT ACTIVATION AND WEIGHT WORD SIZES

Model		6-Bit	8-Bit	10-Bit	float
ResNet-18	Top-1	70.7	72.4	72.8	73.2
ResNet-18	Top-5	89.5	90.7	91.1	91.3
ResNet-50	Top-1	76.9	77.1	77.4	77.8
ResNet-50	Top-5	93.3	93.4	93.6	93.8

TABLE IV
ACCURACY RESULTS (%) FOR RESNET-18 WITH 2-ADD WEIGHT QUANTIZATION AND MULTIPLE FIXED-POINT ACTIVATION WORD SIZES

Acc.	2-Bit	4-Bit	6-Bit	8-Bit	10-Bit	12-Bit	float
Top-1	0.1	65.1	70.3	70.6	70.7	70.7	73.2
Top-5	0.5	86.2	89.7	89.8	89.8	89.8	91.3

TABLE V
ACCURACY RESULTS FOR DIFFERENT QUANTIZATION SCHEMES IN COMPARISON WITH ADDNET [6] FOR RESNET-18 AND RESNET-50 AND MEMORY BANDWIDTH REDUCTIONS COMPARED TO 8 BIT (MEM.) [%]

	ResNet-18					ResNet-50				
	AddNet [6]		Proposed			AddNet [6]		Proposed		
	Top-1	rel.	Top-1	rel.	Mem.	Top-1	rel.	Top-1	rel.	Mem.
float	68.6	100.0	73.2	100.0	-	76.0	100.0	77.8	100.0	-
8 bit	66.0	96.2	72.7	**99.3**	0.0	72.5	95.4	77.4	**99.5**	0.0
6 bit	63.5	92.6	71.7	**98.0**	25.0	69.6	91.6	75.7	**97.3**	25.0
4-Add	66.4	96.8	-	-	0.0	73.3	96.4	-	-	0.0
3-Add	66.0	96.2	72.0	**98.4**	25.0	72.7	95.7	77.1	**99.1**	25.0
2-Add	65.1	94.9	71.0	**97.0**	50.0	72.1	94.9	76.4	**98.2**	49.9
1-Add	-	-	61.9	84.6	75.0	-	-	68.1	87.5	74.9

VI. CONCLUSION

We present a tool flow for automatic layer-wise quantization-aware training (QAT) of DNNs for FPGAs with an adapted metric. The main achievement is a closer matching between DNN computations and FPGA device architecture by using RCCMs avoiding low-bitwidth arithmetic. Our approach combines the optimized layer-wise RCCM selection with an adapted retraining, thus matching a specific weight distribution resulting from our adapted QAT to the FPGA-specific RCCM implementations. To mitigate penalties introduced by RCCM restrictions, we extend the RCCM search space by number with asymmetric coefficient sets. The improved QAT algorithm with an extended search space outperforms previous RCCM concepts and maintains almost floating-point accuracy just using 3-Add RCCMs. This comes with a significant memory bandwidth reduction which can be raised up to 75% with 1-Add RCCMs, however with substantial accuracy losses.

Further advancements might be possible by including a LUT metric into the RCCM selection and to allow different RCCM types on a per-layer basis with a new selection process.

Acknowledgement: This research was supported by NVIDIA's academic hardware grant program as well as by the German Federal Ministry for Digital and Transport (BMDV) as part of the Campus FreeCity project.

REFERENCES

[1] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[2] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, D. Firestone, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "Configurable Clouds," *IEEE Micro*, vol. 37, no. 3, pp. 52–61, 2017.

[3] K. O'Neal and P. Brisk, "Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2018, pp. 763–768, iSSN: 2159-3477.

[4] S. Bavikadi, A. Dhavlle, A. Ganguly, A. Haridass, H. Hendy, C. Merkel, V. J. Reddi, P. R. Sutradhar, A. Joseph, and S. M. P. Dinakarrao, "A Survey on Machine Learning Accelerators and Evolutionary Hardware Platforms," *IEEE Design Test*, pp. 1–1, 2022.

[5] E. Nurvitadhi, S. Subhaschandra, G. Boudoukh, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, and D. Moss, "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*. Monterey, California, USA: ACM Press, 2017, pp. 5–14.

[6] Julian Faraone, Martin Kumm, Martin Hardieck, Peter Zipf, Xueyuan Liu, David Boland, and Philip H. W. Leong, "AddNet: Deep Neural Networks Using FPGA-Optimized Multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–14, 2019.

[7] P. Tummeltshammer, J. C. Hoe, and M. Püschel, "Time-Multiplexed Multiple-Constant Multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1551–1563, Sep. 2007.

[8] A. Pappalardo, "Xilinx/brevitas," 2021. [Online]. Available: https://doi.org/10.5281/zenodo.3333552

[9] D. M. Loroch, F.-J. Pfreundt, N. Wehn, and J. Keuper, "TensorQuant: A Simulation Toolbox for Deep Neural Network Quantization," in *Proceedings of the Machine Learning on HPC Environments*, ser. MLHPC'17, Nov. 2017, pp. 1–8.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: an imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Dec. 2019, no. 721, pp. 8026–8037.

[11] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, vol. abs/1308.3432, 2013. [Online]. Available: http://arxiv.org/abs/1308.3432

[12] J. L. McKinstry, S. K. Esser, R. Appuswamy, D. Bablani, J. V. Arthur, I. B. Yildiz, and D. S. Modha, "Discovering Low-Precision Networks Close to Full-Precision Networks for Efficient Inference," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, Dec. 2019, pp. 6–9.

[13] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," *CoRR*, vol. abs/2103.13630, 2021, arXiv: 2103.13630. [Online]. Available: https://arxiv.org/abs/2103.13630

[14] *7 Series FPGAs Configurable Logic Block User Guide (UG474)*, Xilinx, 2016. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB

[15] G. Brian, G. Dinesh, R. Chirag, and B. Trevor, "Xilinx adaptive compute acceleration platform: Versal architecture," in *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, 2019, pp. 84–93.

[16] F. de Dinechin and B. Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *IEEE Design Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[18] O. Sémery, "Deep learning networks," Oct. 2022, GitHub repository. [Online]. Available: https://github.com/osmr/imgclsmob

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778, iSSN: 1063-6919.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.