MOGNET: A mux-residual quantized network leveraging online-generated weights

Van Thien Nguyen, William Guicquero, Gilles Sicard

MOGNET: A Mux-residual quantized Network leveraging Online-Generated weights

Van Thien Nguyen, William Guicquero and Gilles Sicard
CEA-LETI, F-38000, Grenoble, France
{vanthien.nguyen, william.guicquero, gilles.sicard}@cea.fr

***Abstract*—This paper presents a compact model architecture called MOGNET, compatible with a resource-limited hardware. MOGNET uses a streamlined Convolutional factorization block based on a combination of 2 point-wise (1×1) convolutions with a group-wise convolution in-between. To further limit the overall model size and reduce the on-chip required memory, the second point-wise convolution's parameters are on-line generated by a Cellular Automaton structure. In addition, MOGNET enables the use of low-precision weights and activations, by taking advantage of a Multiplexer mechanism with a proper Bitshift rescaling for integrating residual paths without increasing the hardware-related complexity. To efficiently train this model we also introduce a novel weight ternarization method favoring the balance between quantized levels. Experimental results show that given tiny memory budget (sub-2Mb), MOGNET can achieve higher accuracy with a clear gap up to 1% at a similar or even lower model size compared to recent state-of-the-art methods.**

***Index Terms*—CNN, quantized neural networks, skip connections, channel attention, logic-gated CNN, Cellular Automaton.**

I. Introduction

The successful use of Convolutional Neural Networks (CNNs) in image recognition tasks has been recently accompanied by a considerable increase in model architectures complexity, expanding the number of parameters as well as the computational costs. Unfortunately, this limits the deployment of such network models in embedded systems with limited hardware resources. Therefore, designing lightweight models –regarding memory and computational capabilities– is a challenge to enable accurate inference tasks at the edge. Recent efforts towards alleviating this algorithmic overhead involve several techniques such as efficient model design [1], network quantization [2] and layer inter-connection pruning [3].

Our goal here is to reduce the overall hardware needs required to run a model implemented in resource-constrained devices (*e.g.*, for ASIC design) while still ensuring an acceptable accuracy. Unlike several works focusing on large models to achieve extremely high compression rates [4], [5], we first propose a hardware-compliant model architecture to which we further apply efficient quantization methods.

In this paper, we present the compact MOGNET model architecture which combines:

- quantized residual modules with a Multiplexer-based skip mechanism and,
- a custom factorization of convolution layers that uses on-line generated weights.

Indeed, a Cellular Automaton (CA) is used to automatically generate the weights of a pointwise convolution in each factorized-CNN block, thus reducing parameter-related storage requirements. Moreover, we introduce a novel training framework to obtain the ternary weights in our model which favors the balance between 3 discrete levels.

II. Related works

Residual connections [6] have become important elements of modern CNN architectures, which aim at increasing model expressivity, favoring feature reuse, and alleviating the gradient vanishing in deep CNNs. [7] then incorporates an attention mechanism into the residual learning. SENet [8] proposes a channel attention involving feature aggregation and recablibration stages. MOGNET also employs these aforementioned concepts, however, it mainly focuses on the possible hardware mapping of the model. While previous works perform the residual connections using full-precision, we propose a quantized Multiplexer (MUX) layer with BitShift rescaling allows integrating both an addition connection and a channel-wise attention-like mechanism with a very limited data precision and thus restrained hardware-related costs.

Quantization reduces the precision of weights and activations for low-bitwidth computations. Advanced methods integrate the quantization during training to jointly optimize the quantizer with the quantized weights, under the minimization of the quantization error [9], [10] or the output task loss [11]. In MOGNET, for the sake of versatility, quantizers are regularized to provide balanced quantized outputs.

Efficient architecture design involves the definition of alternative architectures to the hardware-expensive canonical network models. Depthwise Separable Convolution (DSConv [12]) has become a building block in different CNN architectures [13], [1] which performs a depthwise convolution followed by a pointwise (1×1) convolution. [14] revisits DSConv by putting the pointwise (with orthonormal regularization) before the depthwise convolution to cap the redundancy. Grouped convolution is also introduced [15] on the continuum between regular convolution and DSConv. ResNext [16] presents a building block including 2 pointwise layers and a grouped convolution inserted in-between. Our work proposes a factorization similar to the building block of ResNext, without activation inserted in-between layers.

Cellular Automata [17] enable the generation of random states given an initialization and update rule function. Hence,

this repetitive structure is commonly used for generating on-the-fly sets of random vectors [18], or computing in random representations [19]. In MOGNET, we leverage CA to generate part of weight parameters of our model, consequently reducing the overall model memory-related footprint.

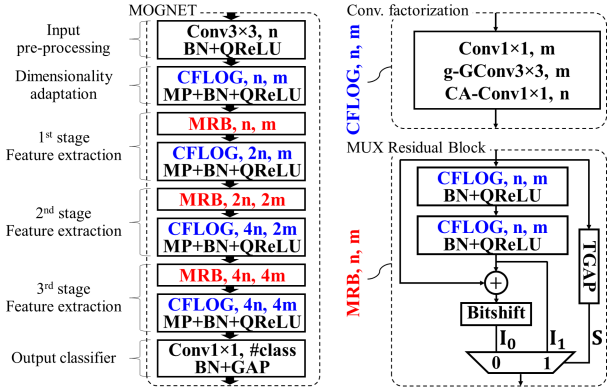


Fig. 1: Top-level architecture description of MOGNET with Convolutional Factorization Leveraging On-line Generated weights (CFLOG) and MUX Residual Block (MRB). The final 1×1 convolution is followed by Batch Normalization (BN) prior to a Global Average Pooling (GAP). Here n, m are the parameters controlling the number of output feature maps and the latent dimension in CFLOG, MP stands for 2×2 Max Pooling and g-GConv is Grouped Convolution with g groups.

III. MOGNET

Figure 1 describe the MOGNET architecture that uses integer-only multiplication-accumulations (MACs) and hardware-compliant operations such as 1-bit Bitshifts and 2-input multiplexers. The following description yet presents MOGNET from its algorithmic view point, *i.e.* with computations done in a real-valued domain but with relevant hardware-equivalent specializations. In this section, we first focus on our custom MUX Residual Block (MRB), then on our convolution layer factorization (CFLOG).

A. MUX residual Block (MRB)

We denote k as the quantization bitwidth of the activations throughout the network. Indeed, a k-bit Quantized Rectified Linear Unit (QReLU) is defined so that for any input, the outputs are in the set $\{0, \frac{1}{2^k-1}, \frac{2}{2^k-1}, ...1\}$:

$$\text{QReLU}(x;k) = \begin{cases} \frac{1}{2^k-1}\lfloor (2^k-1)x \rceil & \text{if } k > 1, \\ \mathbb{1}_{\{x>0\}} & \text{if } k = 1. \end{cases} \quad (1)$$

For the backward pass, we compute the gradient using the Straight-Through-Estimator strategy (STE, [20]) $\frac{\partial \text{QReLU}}{\partial \boldsymbol{x}} = \mathbb{1}_{\{|\boldsymbol{x}|\leq 1\}}$. As the Addition $y = x_1 + x_2$ of two unsigned k-bit activations x_1, x_2 will increase the dynamic range by 1-bit, we make use of the following Bitshift software description inside the MRB, to keep y always at k-bit:

$$\text{Bitshift}(y;k) = \begin{cases} \frac{1}{2^k-1}\lfloor \frac{(2^k-1)y}{2} \rfloor & \text{if } k > 1, \\ \lceil \frac{y}{2} \rceil & \text{if } k = 1. \end{cases} \quad (2)$$

Due to the specific use of this function for the add-type connection, we adopt a completely-passed-through gradient $\frac{\partial \text{Bitshift}}{\partial \boldsymbol{y}} = 1$. For $k > 1$, this rescaling can be implemented by a 1-bit bitshift, while in the specific case of $k = 1$, the combination of the addition and the bitshift can be replaced by an appropriate single **OR** gate. Let us denote $\mathbf{I}_0, \mathbf{I}_1 \in \mathbb{R}^{h\times w\times n}$ as the output of the Bitshift operation and the second QReLU where h, w, n are the height, width and number of channels; $\mathbf{S} \in \{0,1\}^{1\times 1\times n}$ as the binary control signal. The MRB core element is **MUX** which can be mathematically described as:

$$\textbf{MUX}(\mathbf{I}_0, \mathbf{I}_1; \mathbf{S}) = \mathbf{I}_1 \odot \mathbf{S} + \mathbf{I}_0 \odot (\mathbf{1} - \mathbf{S}) \quad (3)$$

where $\odot$ is a channel-wise multiplication. The control signal $\mathbf{S}$ embeds a parameter-free channel attention which consists in a Thresholded Global Average Pooling (TGAP). TGAP simply corresponds to a channel-wise Global Average Pooling (GAP) followed by a binarization $T(x) = \mathbb{1}_{\{x>0.5m\}}$, where m is set to the maximum of the GAP's outputs in the full-precision representation and to 1 in the quantized model which is the maximum possible value of quantized activations. For the hardware deployment, this last operation can be implemented via an integer accumulation followed by an integer-to-integer comparison. This way, the input of each MRB will automatically control the operation of the Multiplexer module in a channel-wise manner. Concretely, MRB will perform the Additional connection for each input feature map that is dominated by small values. Otherwise, the MRB will simply keep the straightforward output of the second QReLU. One interesting aspect of this MUX-skip connection is that it favors the balance between the number of large-valued data with respect to the number of small-valued data throughout the networks, this without any other regularization strategy.

B. Convolution factorization leveraging CA-generated weights

To further reduce the on-chip memory and the computational complexity of the model, we replace all regular convolutions (except the first and the last layers, cf. Fig. 1) by a light-weight factorization consisting of 2 pointwise layers and a grouped convolution (Fig. 2), namely CFLOG. Unlike the building block in ResNext [16], we do not use any non-linearity (*e.g.* normalization, activation) between these layers. Moreover, to further reduce the model size, the last pointwise convolution's weights are fixed during training and generated in real-time by a CA given a certain seed. The first pointwise convolution embeds the input feature into low-dimension $m < C_i$, the number of input channels. The grouped conv layer performs g groups of convolutions and also outputs m feature maps. Finally, these feature maps are sequentially projected back to a high-dimensional space of C_o channels ($C_o = n$ in CFLOG, n, m) thanks to a CA-generated kernel. As depicted in Fig. 2, this kernel is formed by concatenating all states obtained when evolving a C_o-cell CA during m update states. In this work, we consider Wolfram's rule 30 for the local evolution function between states. We choose $m = \frac{C_i}{2}$ that gives the following compression rate (CR) between the

number of trainable parameters ($\#pr$) of CFLOG and that of the regular convolution:

$$\text{CR} = \frac{C_i m + \frac{3^2 m^2}{g}}{3^2 C_i C_o} = \frac{C_i}{C_o}\left(\frac{1}{18} + \frac{1}{4g}\right) \tag{4}$$

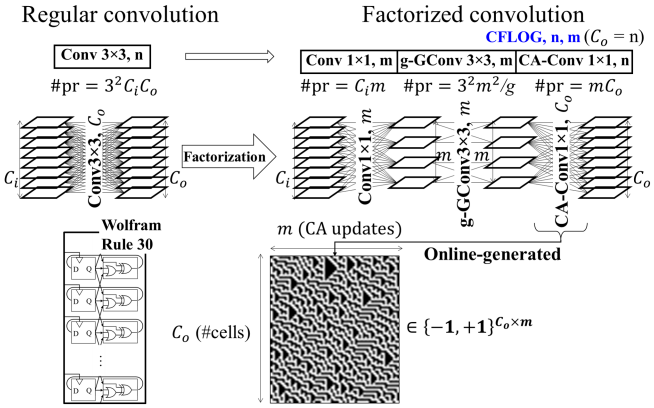


Fig. 2: CFLOG description with CA-generated weights.

C. Balanced Ternary Quantization (BTQ)

In order to drastically compress the model, we binarize [21] all learnable pointwise and ternarize all other layers' weights. To obtain the ternarized parameters, we introduce a novel quantization-aware training scheme which favors the balance between 3 discrete levels. The ternary mapping $q : \mathbb{R} \rightarrow \{-1, 0, +1\}$ is applied to the real-valued proxy weights $\boldsymbol{w}$,

$$q(w; s) = \text{Clip}\left(\left\lfloor \frac{w}{s} \right\rceil, -1, 1\right), \tag{5}$$

where s is the step size parameter. We still adopt the STE gradient $\frac{\partial q}{\partial \boldsymbol{w}} = \mathbb{1}_{\{|\boldsymbol{w}| \leq 1\}}$. In Fig. 3 we denote q_1 and q_2 as the **tertiles** of the proxy weight's histogram. Observing that the proxy weights distribution may change during the training, but the median value usually stays around zero, we assume that q_1 and q_2 are symmetrically distributed, *i.e.* $q_1 \approx -q_2$ with $q_2 > 0$. Therefore, to equi-distribute the quantized weights, we can automatically update the step size s at the beginning of each epoch based on q_1 and q_2. Concretely, we expect that the sum of the absolute value of the tertiles (q_1, q_2) is approximately equal to that of the thresholds $(-\frac{s}{2}, \frac{s}{2})$, therefore we have:

$$s = |q_1| + |q_2|. \tag{6}$$

The algorithm for training BTQ is detailed in Algorithm 1, in which the step size update is described in lines $2 - 5$.

IV. Experiments

A. Experimental settings

We implemented all the proposed elements using the TensorFlow [22] library and CellPylib [23] package. The quantized models are first initialized from their full-precision counterparts being trained on CIFAR-10 and CIFAR-100 [24] datasets from scratch, where ReLU and Linear activations replace our QReLU and BitShift. Then, we train the quantized

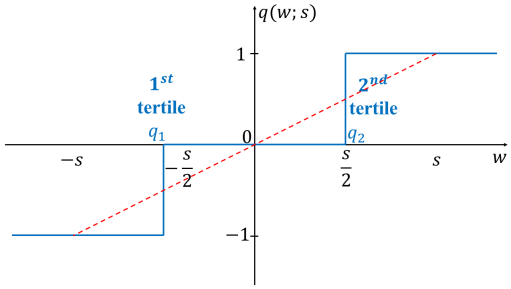


Fig. 3: Balanced ternary quantization with histogram bin equalization when 2 tertiles (q_1, q_2) are symmetrical and coincide with the quantization thresholds $(-\frac{s}{2}, \frac{s}{2})$.

Algorithm 1 Training MOGNET with BTQ

Input: Initial proxy weights $\{\mathbf{W}_l\}_{l=1}^{L}$ and training dataset
Output: Optimized $\{\mathbf{W}_l\}_{l=1}^{L}$, $\{s_l\}_{l=1}^{L}$
// B, T, L: #batches, #epochs, #layers using BTQ
// $\mathbf{W}_l$: full-precision proxy weights of the l^{th} layer,
// s_l: quantization step size used at the l^{th} layer,
1: **for** $t = 1$ to T **do**
2: **for** $l = 1$ to L **do**
3: Find 2 tertiles q_1 and q_2 of layer l
4: Compute and update s_l (Eq. 6)
5: **end for**
6: **for** $b = 1$ to B **do**
7: Forward pass using $\{q(\mathbf{W}_l; s_l)\}_{l=1}^{L}$ (Eq. 5)
8: Backward pass and update $\{\mathbf{W}_l\}_{l=1}^{L}$
9: **end for**
10: **end for**

models through a 2-stage procedure: first train quantized weights with full-precision activations and second, fine-tune the quantized weights with all quantized activations. We apply the simple data augmentation scheme for training: random crop from all-sided 4-pixel padded images combined with random horizontal flips. Table I details the training and optimization setting used to derive our experimental results.

TABLE I: Training and optimization settings

Dataset	CIFAR-10	CIFAR-100
Optimizer	Adam [25] ($\beta_1 = 0.9, \beta_2 = 0.999$)	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Initial learning rate (LR)	10^{-3}	10^{-3}
Batch size	50	50
First stage epoch	180	250
First stage LR schedule	Exponentially decay after 120-th epoch	Exponentially decay after 150-th epoch
Second stage epoch	150	250
Second stage LR schedule	Exponentially decay after 80-th epoch	Exponentially decay after 150-th epoch
Rate of LR decay	0.9	0.9

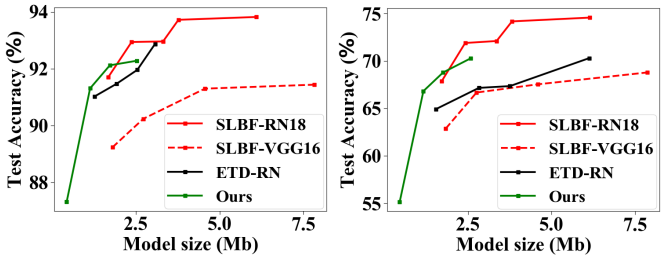
B. Experimental results

We evaluate the performance of our models in comparison with recent state-of-the-art model compression techniques: Stacking Low-dimensional Binary Filters (SLBF [26]) on

ResNet (RN)-18 and VGG-16 [27]; Efficient Tensor Decomposition (ETD [28]) on RN-20 and RN-32. Table II reports the model size, the activation precision as well as the accuracy of different methods and models. It demonstrates that for $n = 128$ and $g = 4$, MOGNET achieves the highest accuracy level on CIFAR-10, while having lower model size and 3-bit only activations. Moreover, at the same configuration, MOGNET outperforms other methods on CIFAR-100 with a clear gap of nearly 1% ($67.89 \rightarrow 68.80\%$) at the similar weight-related memory. We can mention the impact of the hyperparameter k on the model performance, with a significant degradation when decreasing the activation precision to 1-bit or 2-bit. Figure 4 additionally reports accuracy versus model size curves of various considered models/hyperparameters. On both two datasets, MOGNET stays in the optimal top-left zone implying low on-chip memory requirements with high accuracy. However, when increasing the model size ($n > 128$), MOGNET curves fall under that of SLBF-RN18. This last result means that MOGNET, with its limited depth, is more relevant to target extremely low-sized (< 2Mb) models.

TABLE II: Comparison of different network compression methods on CIFAR-10 and CIFAR-100.

Method Model	Activation Bitwidth (k)	CIFAR-10 Model size (Mb)	CIFAR-10 Acc (%)	CIFAR-100 Model size (Mb)	CIFAR-100 Acc (%)
SLBF [26]					
RN-18	32	1.67	91.70	1.72	67.89
VGG-16	32	1.84	89.24	1.89	62.88
ETD [28]					
RN-20	32	1.94	91.47	3.80	67.36
RN-32	32	2.54	91.96	2.83	67.17
Ours (n=128, g=8)	1	1.13	87.60	1.22	59.30
	2		90.81		65.88
	3		91.31		66.83
Ours (n=128, g=4)	1	**1.72**	88.99	**1.76**	61.27
	2		91.16		66.55
	3		**92.12**		**68.80**



(a) CIFAR-10. (b) CIFAR-100.

Fig. 4: Test accuracy of different compression method-model couplings. Our models are with 3-b activations.

V. Conclusion

This work introduces a novel hardware-compliant quantized model architecture called MOGNET, which integrates a custom Multiplexer mechanism and a lightweight convolution factorization that leverages Cellular Automaton-generated weights, in order to limit the hardware memory needs. We empirically show that our method can achieve better accuracy with lower model size than previous works, in particular for a tiny memory budget, while limiting the digital dynamic range using 3-bit quantized activations for integer-only MACs.

References

[1] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.

[2] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *ArXiv*, vol. abs/1609.07061, 2017.

[3] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, 2015.

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[5] D. Oktay, J. Ballé, S. Singh, and A. Shrivastava, "Scalable model compression by entropy penalized reparameterization," in *ICLR*, 2020.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[7] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *CVPR*, 2017.

[8] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *CVPR*, 2018.

[9] F. Li and B. Liu, "Ternary weight networks," *ArXiv*, vol. abs/1605.04711, 2016.

[10] X. Zhao, Y. Wang, X. Cai, C. Liu, and L. Zhang, "Linear symmetric quantization of neural networks for low-precision integer hardware," in *ICLR*, 2020.

[11] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *ICLR*, 2020.

[12] L. Sifret, *Rigid-Motion Scattering For Image Classification*. Ecole Polytechnique, CMAP PhD thesis, 2014.

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[14] D. Haase and M. Amthor, "Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets," in *CVPR*, 2020.

[15] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *CVPR*, 2018.

[16] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.

[17] S. Wolfram, *A New Kind of Science*. Champaign, Ilinois, USA: Wolfram Media Inc., 2002.

[18] J. Liu and Q. Sun, "Chaotic cellular automaton for generating measurement matrix used in cs coding," *IET Signal Process.*, vol. 11, 2017.

[19] Ö. Yılmaz, "Reservoir computing using cellular automata," *ArXiv*, vol. abs/1410.0162, 2014.

[20] Y. Bengio, N. Léonard, and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *arXiv:1308.3432 [cs]*, Aug. 2013.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *NeurIPS*, 2016.

[22] M. A. et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *ArXiv*, vol. abs/1603.04467, 2016.

[23] L. M. Antunes, "Cellpylib: A python library for working with cellular automata," *Journal of Open Source Software*, vol. 6, no. 67, 2021.

[24] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[26] W. Lan and L. Lan, "Compressing deep convolutional neural networks by stacking low-dimensional binary convolution filters," in *AAAI*, 2021.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[28] M. Yin, Y. Sui, S. Liao, and B. Yuan, "Towards efficient tensor decomposition-based DNN model compression with optimization framework," in *CVPR*, 2021.