**Les Cahiers du GERAD** ISSN: 0711–2440

# Pruning for efficient hardware implementations of deep neural networks

G. Boukli Hacene, V. Gripon,
M. Arzel, N. Farrugia, Y. Bengio

G–2020–23–EIW02

April 2020

# Pruning for efficient hardware implementations of deep neural networks

**Ghouthi Boukli Hacene**

**Vincent Gripon**

**Matthieu Arzel**

**Nicolas Farrugia**

**Yoshua Bengio**

*MILA/IMT Atlantique, Montréal (Québec), Canada, H2S 3H1*

bouklihg@mila.quebec

**April 2020**
**Les Cahiers du GERAD**
**G–2020–23–EIW02**

**Abstract:** *Convolutional Neural Networks (CNNs) are state-of-the-art in numerous computer vision tasks such as object classification and detection. However, the large amount of parameters they contain leads to a high computational complexity and strongly limits their usability in budget-constrained mobile devices. In this paper, we propose a combination of a pruning technique and a quantization scheme that reduces complexity and memory of convolutional layers of CNNs, by replacing the complex convolutional operation by a low-cost multiplexer. We perform experiments on CIFAR10, CIFAR100, and SVHN and show that the proposed method achieves almost state-of-the-art accuracy, while drastically reducing the computational and memory footprint. We also propose an efficient hardware architecture to accelerate inference, which works as a pipeline and accommodates multiple layers working at the same time. In contrast with most proposed approaches that have used external memory or software defined memory controllers, our work is based on algorithmic optimization and full-hardware design.*

## 1 Introduction

For the past few years, Deep Neural Networks (DNNs), and especially Convolutional Neural Networks (CNNs) [7], have received considerable attention thanks to their remarkable accuracy in computer vision tasks [6, 8, 2, 9]. However, the need for intensive computations and memory leaded to the fact most DNN implementations are based on GPUs. Consequently, providing efficient hardware implementations is a very active and prospective field of research. Therefore, the deployment of CNNs in embedded systems is complex and potentially blocking for many applications.

In this paper, we propose to combine Shift Attention Layer (SAL) [3] a substitution to convolutional layers, in which the complex convolution operation is replaced by a shift operation followed by a multiplication with binary quantization of weights using Binary Weight Network (BWN), resulting in very lightweight DNNs in which complex convolution operations are replaced by low cost multiplexers that considerably eases hardware implementation on FPGA. We show in the following that such a combination approaches state-of-art accuracy while reducing computational and memory footprint. We also propose a hardware architecture in which we consider all processing blocks, memory blocks and controllers required, and implement such an architecture which uses very few resources and computational power on an FPGA, without using any external resources. This implementation can compute more than one layer at a time and uses a simple multiplexer to replace convolutional operations and process data through DNN layers. As such, it provides significantly smaller latency than existing counterparts, as shown in Section 3.

## 2 Shift layers and Shift Attention Layers

Let us denote by $\mathbf{x}$ (resp. $\mathbf{y}$ or $\mathbf{w}$) the input (resp. output or kernel) tensor of a given convolutional layer. We index $\mathbf{x}$ (resp. $\mathbf{y}$) using three indices $i$, $j$, $k$ (resp. $\ell$), where $0 \leq i < i_{\max}$ and $0 \leq j < j_{\max}$ correspond to 2D coordinates and $0 \leq k < k_{\max}$ (rsp. $0 \leq \ell < \ell_{\max}$) indexes a feature map. Similarly, we index $\mathbf{w}$ using four indices: $0 \leq \iota \leq \iota_{\max}$ and $0 \leq \lambda \leq \lambda_{\max}$ correspond to 2D coordinates, and $k$ and $\ell$ are as introduced above. So, an element of the input tensor is written $x_{i,j,k}$, an element of the kernel tensor is written $w_{\iota,\lambda,k,l}$ and an element of the output tensor is written $y_{i,j,l}$.

To obtain a Shift Layer (SL) instead of a Convolutional Layer (CL), the authors in [11] propose to remove most of the connections in each slice $\mathbf{w}_{\cdot,\cdot,k,\ell}$ of the kernel tensor at the initialisation. The connections to be kept are chosen according to a deterministic rule agnostic of the initialization and of the training dataset. Namely, the authors choose to only keep the connections $w_{\iota,\lambda,k,\ell}$ for which

$$\iota + \lambda\iota_{\max} = k \pmod{\iota_{\max}\lambda_{\max}}. \tag{1}$$

When considering $3 \times 3$ kernels for example, 89% of the connections are pruned in the convolutional layer. Then, the training process is performed on remaining connections, disregarding the other ones.

Using such a configuration, all connections in a $3\times3$ kernel are pruned but one, and thus the convolution of each slice of the kernel tensor is replaced by a simple multiplication.

Contrary to SL where the kept connection is predetermined, SAL uses an attention mechanism [10] that selects the most relevant connection for each kernel and prunes the others. This process is performed during training, so that at the end of the training phase, the network configuration corresponds exactly to one obtained using SL.

To further benefit from the reduced complexity of the SAL method, we combine it with a weight binarization method such as binary weight network (BWN). Once remaining connections have been binarized, it is possible to replace the multiplication operation by a multiplexer, and thus, such a combination requires only low cost multiplexers to process data during inference.

### Results

To evaluate the performance of our proposed combination, we use the CIFAR10 dataset. We compare various modern CNN architectures such as Resnet [4], Wide-Resnet [12] and Densenet [5]. Note that these architectures contain $1 \times 1$ and $3 \times 3$ convolutional kernels only. Thus we apply the proposed method on the $3 \times 3$ kernels.

We report in Table 1 the obtained results when using Equation (1) to remove kernels connections (SL), when applying SAL, and when combining SAL with BWN, and compare the accuracy obtained with baseline architectures. Note that BWN offers a 32 compression factor in terms of memory used, and SL or SAL method roughly multiply this factor by 9, achieving an almost 300 factor compression in total. Interstingly, we observe that when using SAL with BWN, the obtained accuracy remains at most 1% away to that of the baseline. We also perform experiments on SVHN (resp. CIFAR100) on Resnet18 and obtain 97%/96% (resp. 78%/75.2%) accuracy for Full-precision/SAL+BWN.

**Table 1: Comparison of accuracy between baseline architectures, pruned ones, binarized ones, and the proposed method on CIFAR10.**

| | Resnet18 | Resnet34 | WideResnet-28-10 | Densenet121 |
|---|---|---|---|---|
| Full-precision | 94.5% | 95% | 96.2% | 95% |
| SL | 93.5% | 93.8% | 95% | 94.3% |
| SAL | 94.2% | 94.9% | 96% | 94.8% |
| SAL + BWN | 93.5% | 94.6% | 95.4% | 94.6% |

## 3 Hardware implementation

The processing unit uses **X** (a feature vector, corrsponding to a row in a feature map) and a vector **W** made of $P$ values coded on 1 bit each corresponding to weights. It thus computes in parallel $P$ feature vectors (cf. Figure 1). The First-Input signal (FI) is set to 1 when the first feature vector is read from the BRAM to initialise registers by 0. To compute each feature vector $p$ where $1 \leq p \leq P$, we use the corresponding $W_p$ to add either **X** or **-X** to the content of register $p$. Once all input feature vectors have been read from the BRAM , the signal `Enable_s` is set to 1, and the content of registers is written one by one into the BRAM of the next layer. At the end of this process, the `Itter_done` signal is set to 1 in the processing unit block, so new data can be read to process other feature vectors.

To compute inference, $k_{max}j_{max}$ clock cycles (CCs) are required to copy all contents from a first layer's BRAM to a second layer's BRAM, $j_{max}k_{max}\ell_{max}/P$ CCs to compute all output feature vectors of one layer, and $j_{max}\ell_{max}$ CCs to write all computed feature vectors into the BRAM of a third layer. Thus the total number of CCs required is:

$$CCs = j_{max}k_{max} + \frac{j_{max}k_{max}\ell_{max}}{P} + j_{max}\ell_{max}. \tag{2}$$

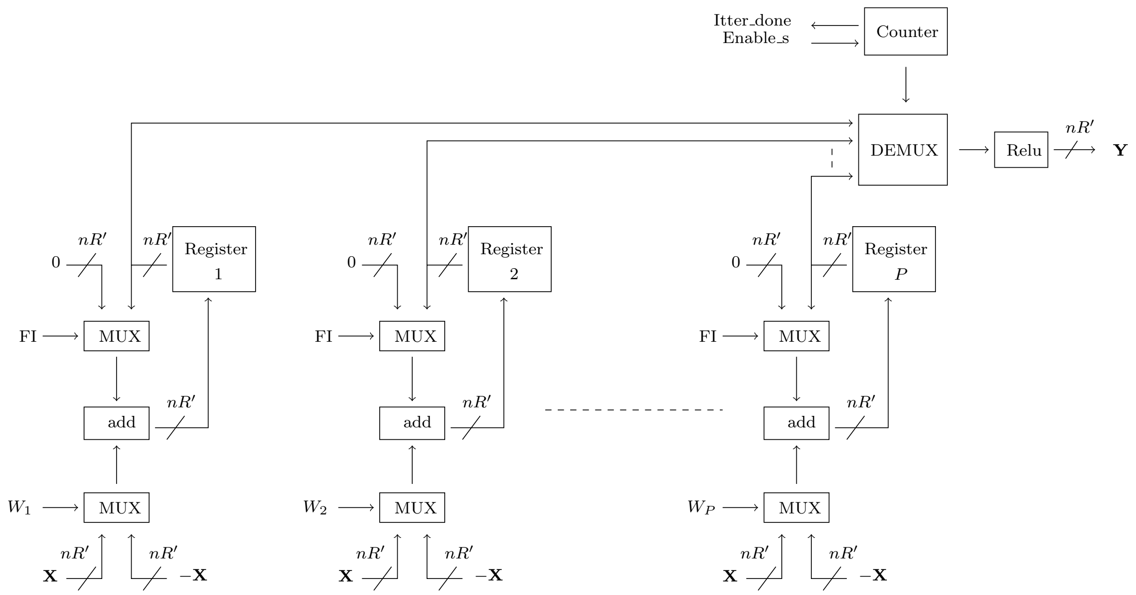

**Figure 1: Hardware architecture of a processing unit block.**

When comparing the total number of CCs of the proposed method with those obtained in [1], that are introduced by the following equation:

$$CCs = \frac{3j_{max}^2 k_{max}\ell_{max}}{P}, \tag{3}$$

we observe that the proposed architecture is $3j_{max}$ faster than [1], which can be significant when $j_{max}$ is big. For instance with the CIFAR10 dataset, at the input layer of a CNN $j_{max} = 32$, and thus the proposed method is 96 times faster. In addition it is a pipeline architecture, so it can be $3Lj_{max}$ faster where $L$ is the total number of layer blocks that fit in an FPGA.

## Hardware results

We implemented one/few layers of Resnet18 on Xilinx Ultra Scale Vu13p (xcvu13p-figd2104-1-e) FPGA (c.f. Table 2). The implemented layers are arranged in a pipeline, and their functionality has been verified comparing the output of each layer block with the ones obtained by software simulation over a batch of examples.

**Table 2: FPGA results for the proposed architecture on vu13p (xcvu13p-figd2104-1-e).**

| | P | LUT | FF | BRAMs | Frequency | Processing outflow | Power |
|---|---|---|---|---|---|---|---|
| Conv64 − 64 | 16 | 22424 | 22424 | 114 | 240MHz | 19230 images/s | 3.7W |
| 4×Conv64 − 64 | 16 | 89746 | 75235 | 456 | 240MHz | 19230 images/s | 6.5W |
| 3×Conv128 − 128 | 64 | 134090 | 102552 | 171 | 240MHz | 29069 images/s | 7.8W |
| 3×Conv256 − 256 | 128 | 154599 | 102723 | 87 | 218MHz | 26595 images/s | 7.8W |
| 3×Conv512 − 512 | 128 | 132155 | 52151 | 45 | 208MHz | 16949 images/s | 7.9W |

## 4 Conclusion

In this paper, we proposed to reconsider the hardware implementation and acceleration of DNNs on limited resources embedded systems such as FPGA. We proposed to use lightweight DNNs architectures based on shift attention layers, and to combine them with weight binarization to reduce both complexity and memory usage. The resulting architecture almost matches the accuracy of considered baselines and only requires multiplexers, easing hardware implementation.

We implemented the proposed scheme using a low cost hardware architecture in which complex convolution operations are replaced by multiplexers. Thus, we were able to implement a considerable part of a complex CNNs (Resnet18). Moreover, the architecture only consumes a few watts and does not use any external ressources, making it a good solution for embedded applications.

## References

[1] Arash Ardakani, Carlo Condo, and Warren J Gross. A convolutional accelerator for neural networks with binary weights. In Circuits and Systems (ISCAS), 2018 IEEE International Symposium on, pages 1–5. IEEE, 2018.

[2] Benjamin Graham. Fractional max-pooling. CoRR, abs/1412.6071, 2014.

[3] Ghouthi Boukli Hacene, Carlos Lassance, Vincent Gripon, Matthieu Courbariaux, and Yoshua Bengio. Attention based pruning for shift networks. arXiv preprint arXiv:1905.12300, 2019.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[5] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In CVPR, volume 1, page 3, 2017.

[6] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $< 0.5$ mb model size. arXiv preprint arXiv:1602.07360, 2016.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, pages 5998–6008, 2017.

[11] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. arXiv preprint arXiv:1711.08141, 2017.

[12] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. arXiv preprint arXiv:1605.07146, 2016.