Vaasan yliopisto
UNIVERSITY OF VAASA
OSUVA Open Science

This is a self-archived – parallel published version of this article in the publication archive of the University of Vaasa. It might differ from the original.

Neural Network-based Vehicle Image Classification for IoT Devices

Author(s): Payvar, Saman; Khan, Mir; Stahl, Rafael; Mueller-Gritschneder, Daniel; Boutellier, Jani

Title: Neural Network-based Vehicle Image Classification for IoT Devices

Year: 2019

Version: Final draft (post print, aam, accepted manuscript)

Copyright ©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Please cite the original version:

Payvar, S., Khan, M., Stahl, R., Mueller-Gritschneder, D., & Boutellier, J., (2019). Neural Network-based Vehicle Image Classification for IoT Devices. IEEE International Workshop on Signal Processing Systems (SiPS), Nanjing, China, 2019, pp. 148-153. https://doi.org/10.1109/SiPS47522.2019.9020464

Neural Network-based Vehicle Image Classification for IoT Devices

Saman Payvar
Unit of Computing Sciences
Tampere University
Tampere, Finland
saman.payvar@tuni.fi

Mir Khan
Unit of Computing Sciences
Tampere University
Tampere, Finland
mir.markhan@tuni.fi

Rafael Stahl
Chair of Electronic Design Automation
Technical University of Munich
Munich, Germany
r.stahl@tum.de

Daniel Mueller-Gritschneder
Chair of Electronic Design Automation
Technical University of Munich
Munich, Germany
daniel.mueller@tum.de

Jani Boutellier
Tampere University /
University of Vaasa
Finland
jani.boutellier@tuni.fi

***Abstract*—Convolutional Neural Networks (CNNs) have previously provided unforeseen results in automatic image analysis and interpretation, an area which has numerous applications in both consumer electronics and industry. However, the signal processing related to CNNs is computationally very demanding, which has prohibited their use in the smallest embedded computing platforms, to which many Internet of Things (IoT) devices belong. Fortunately, in the recent years researchers have developed many approaches for optimizing the performance and for shrinking the memory footprint of CNNs. This paper presents a neural-network-based image classifier that has been trained to classify vehicle images into four different classes. The neural network is optimized by a technique called binarization, and the resulting binarized network is placed to an IoT-class processor core for execution. Binarization reduces the memory footprint of the CNN by around 95% and increases performance by more than 6×. Furthermore, we show that by utilizing a custom instruction 'popcount' of the processor, the performance of the binarized vehicle classifier can still be increased by more than 2×, making the CNN-based image classifier suitable for the smallest embedded processors.**

***Index Terms*—model compression, convolutional neural networks, image classification, internet-of-things**

I. INTRODUCTION

Convolutional neural networks (CNNs) have enabled a significant advance in automatic image analysis, such as image classification [1], image segmentation [2], image captioning [3] and object detection [4]. Unfortunately, up to recently the computational requirements of CNNs have restricted their use to server or desktop class computers, although their deployment to *edge devices* could open up a variety of new applications [5]. In the Internet-of-Things (IoT), the network edge refers to devices that are within immediate connection to sensors that provide input data for the whole IoT system. Such an edge device can be a smartphone [6], or a tiny sensor node commonly equipped with less than a megabyte of RAM [7].

A CNN consists of a sequence of *layers*, of which the most common types are *fully-connected layers* and *convolutional layers*. Once a CNN has been trained [8], e.g. for image classification, the *parameters* and *weights* of the layers are fixed for deployment to a target device. On the target device, the process that evaluates given input data is called *inference*, where the input data flows through the layers of the CNN, providing the requested output (e.g. classification result) from the last layer.

In terms of computation, convolutional layers consist of repeated 2D convolutions, where the input data of the layer is convolved by 2D kernels with common sizes of 5×5, 3×3 or 1×1 [9]. The computational effort of convolutional layers grows rapidly as the size of input images or kernels grows [10]. However, it has been well-known for some time that 2D convolution can also be interpreted and computed as a 2D matrix multiplication [11]. The inference of a fully-connected layer is also commonly performed by 2D matrix multiplication.

Optimization of CNN processing can be performed by optimizing software, hardware, or both [12]. Examples for software-based optimizations are model compression [9][13] or reduction of arithmetic precision [14][12]. Software-based optimizations that target convolutional layers include separable convolution [15] and depthwise convolution [16], whereas fully-connected layers can be optimized by weight pruning [13]. All of these optimizations have some negative impact on the CNN accuracy.

Reduction of arithmetic precision, on the other hand, is not limited to separate types of layers, but can be applied to the whole CNN. Arithmetic precision can be reduced from floating-point to, e.g., 16-bit fixed point [12] with minimal degradation of CNN (classification) accuracy, or by extreme quantization down to two [17] bits or one bit [18][14] of weight precision. When the precision of weights (and possibly also input data) is reduced to a single bit, the CNN is *binarized*. Binarization dramatically reduces the memory footprint of a CNN, as the original weights, which are normally expressed in 32-bit floating point, can be represented with a single bit. This evidently has an impact on the CNN's accuracy [18]. However, besides shrinking the size of the

TABLE I
RELATED NEURAL NETWORK OPTIMIZATION WORKS

Work	Type	Optimization	Platform
Courbariaux et al. [18]	SW only	Binarization (fc layers only)	NVidia GPU
Rastegari et al. [24]	SW only	Binarization (conv and fc layers)	64-bit CPU
Khan et al. [14]	SW only	Binarization (conv and fc layers)	NVidia and OpenCL GPUs
ESPRESSO [25]	SW only	Binarization (conv and fc layers)	NVidia GPU, CPU
Park et al. [26]	HW SW	Zero skipping, Data reuse (conv layers only)	Nvidia GPU, GPU simulation
Conti et al. [27]	HW SW	Binarization (conv and fc layers)	HW accelerator for MCUs
Proposed	HW SW	Binarization (conv and fc layers)	RISC-V MCU (simulation)

network, binarization also enables CNN inference on devices that have no support for floating-point arithmetic, such as microcontrollers and FPGAs [19].

This paper presents a CNN for vehicle image classification [20] that has been binarized including the weights of all layers, as well as the input data, following the principles of our recent work [14]. However, unlike our recent work that concentrated on CNN inference on graphics processing units, in this paper we focus on microcontroller-class devices that can be found on edge nodes of an IoT system. As the target microcontroller, we have selected PULPino [21], which is based on the open-source instruction-set architecture RISC-V [22], which is gaining interest in both academia and industry.

The contributions of this paper are as follows:

- Performance and memory footprint measurements of our binarized CNN-based image classifier on a RISC-V microcontroller, and
- Optimization of binarized CNN computations by the custom instruction 'popcount' found in a proposal for RISC-V instruction set extensions [23].

The structure of this paper is as follows: Section II introduces other works related to optimization of CNNs; Section III describes the PULPino microcontroller that we use as the target device for our image classifier; Section IV covers the structure and binarization process of our CNN; Section V presents our experimental results, and Section VI concludes the paper.

II. RELATED WORK

This section describes previous works related to acceleration of CNNs, some also considering acceleration by hardware. Table I presents a summary of these works and the target platforms they consider.

Binarized neural networks (BNN) were originally introduced in [18]: network weights and activations are restricted to $+1$ and -1, which enables replacing multiplications and additions with bit-wise operations. Experiments have been performed on MNIST and CIFAR-10 datasets. The authors demonstrate a speedup of $7\times$ for a multi-layer perceptron network trained for MNIST handwritten digit classification. Experimental results are limited to GPU acceleration of binarized fully-connected layers.

Somewhat later the binarization optimization was extended to the large-scale ImageNet image classification challenge [24]. The authors of [24] concentrate on CPU targets and report up to $58\times$ execution time reduction on 64-bit CPUs for binarized convolution and fully-connected layers. Also, the authors claim an accuracy improvement of 16% compared to [18] in the ImageNet top-1 classification challenge.

Our previous work [14] was among the first ones to present GPU acceleration of both binarized convolution and fully-connected layers. Experimental results are presented for two mobile GPUs (NVidia Jetson and ARM Mali-T860), as well as for a desktop GPU (NVidia GTX1080). Layer implementations have been written from scratch in OpenCL and CUDA and made available open source. Additionally, the accuracy impact of various input image binarization approaches are analyzed.

In [25] a self-contained library ESPRESSO for binarized neural networks is presented. The library provides layer implementations in C and CUDA for both CPU and NVidia GPU targets. ESPRESSO [25] uses an optimization called *unrolling* (similar to *im2col* used in our previous work [14] and the proposed work) for reshaping tensors prior to computing convolution.

Optimization of CNN convolution operations is studied in [26]. The authors have observed that Winograd convolutions can involve a high number of multiplications by zero, especially if weight pruning (see, e.g. [13]) has been applied. This redundancy is avoided by skipping zero weights by a software-only and by a hardware-assisted approach. Additionally, the authors present a data reuse approach for reducing the number of additions. Both optimizations target NVidia GPUs.

In [27] the XNOR Neural Engine (XNE) is presented, a hardware accelerator for binary neural networks to be closely coupled with an MCU (micro controller unit) system. The XNE is capable of executing both binarized convolutional and fully-connected layers. The authors provide post-layout results where the accelerator has been placed on the same chip and same clock domain with a RISC-V microcontroller that acts as the host processor for the accelerator.

The proposed work is similar to the work of Conti et al. [27] in the sense that both consider an IoT edge computing scenario, build on binarized CNNs, and consider RISC-V MCU cores. However, a substantial difference is that the XNE accelerator of [27] is a dedicated datapath for CNNs next to the MCU core, whereas our proposed solution builds on a basic microcontroller architecture with just one custom processor instruction ('popcount') for accelerating BNNs. Evidently, the specialized circuit of [27] can achieve much higher energy efficiency than our proposed solution, whereas our solution only requires a tiny modification to a basic RISC-V MCU system, and otherwise remains very generic and capable of accelerating other types of applications as well.

Fig. 1. From left to right, a 'bus', 'normal car', 'truck', and a 'van'.

III. THE PULPINO RISC-V PROCESSOR FOR IOT APPLICATIONS

RISC-V is an open source instruction set architecture (ISA) that is gaining interest in both academia and industry [22]. The ISA is open and standardized, such that it is free to use for both academia and industry. To promote adoption of the new ISA, another goal was to design a modern ISA: it is designed in a modular way by providing a small base instruction set with optional extensions. Additionally, certain instruction opcodes are reserved for custom extensions. This flexibility allows to design RISC-V processors that are customized for special workloads, which makes the ISA interesting for specialized IoT devices.

While the open standard is just referring to the ISA itself and not any micro-architecture, the community around RISC-V has provided many open-source cores. An important motivation for open hardware is security, especially with recent micro-architecture bugs Spectre and Meltdown appearing in popular media [28][29]. Kerckhoff's principle and a long history of research suggests that open systems provide certain advantages over closed systems in terms of security [30][31][32].

The Parallel Ultra-Low-Power (PULP) project has developed several RISC-V-based microcontrollers that are suitable for IoT applications [21]. The PULPino is particularly suited for low cost, low power tasks, because it is a simple in-order single-core microcontroller with many configuration options. Due to these advantages, the custom processor used in this work was derived from the PULPino-based SoC (System-on-Chip).

IV. NEURAL NETWORK DESIGN

A. Network for Vehicle Classification

The neural network model we use is that of the vehicle classifier network presented in [20]. The network has five layers in total, starting with two convolutional layers, each one with 32 output feature maps, and kernel sizes 5×5. Each of the convolutional layers is followed by a 2×2 maxpooling operation. The second convolutional layer is followed by three fully-connected layers. The first fully-connected layer (the 3rd layer in the network) has 100 neurons, resulting in the shape 24×24×32×100. The two layers that follow have shapes 100×100, and 100×4, in that order.

The dataset we use for training the network has 6555 photos of vehicles from four categories: *bus*, *normalcar*, *truck*, and *van*. Each vehicle image is a full-color image of size 96×96. Example images from each class in the data set are shown in Fig. 1. We split the data into a training set (80%), validation set (10%) and a test set (10%). Our test-set accuracy reports are the recorded accuracy reports that correspond to the best validation set accuracy.

B. Neural Network Binarization

We implement a binarized version of the vehicle classifier network introduced in [20] reducing the precision of CNN weights and their activations to 1-bit. This concept was first introduced in [18], with reports of substantial reductions of model execution time and size. In this work, we replace all ReLU activations in the network with the sign function, which is given as

$$sign(x) = \begin{cases} -1 & \text{if } x \leq 0 \\ +1 & \text{if } x > 0 \end{cases} \tag{1}$$

We binarize the weights of the network using the $sign$ function as well. During training, the gradient of *sign* activations are explicitly defined to be the identity function in the backward pass so that $\frac{\partial sign(x)}{\partial x} = x$. The full-precision version of the network (non-binarized) is trained using the RMSprop optimizer, and the binarized version is trained with the ADAM optimizer. For the binarized version of the network, only the binarized weights, where all have a value of either -1 or $+1$, are used for inference on the target device. The network is trained from scratch using binarization in a separate training process. It would also have been possible to quantize the network to ternary values [17] (or even higher 8- or 16-bit precision), but that would have multiplied the memory footprint of the solution compared to binarization.

We use the terms *packing* or *bit-packing* to denote the encapsulation of an array of 1-bit values ($+1$'s and -1's) into one 32-bit unsigned integer. For example, if we wish to *pack* a vector $\mathbf{x} \in \{-1, +1\}^{32}$, its *packed* representation, x^p, is given by

$$x^p = \sum_{i=0}^{31} (x_i + 1) 2^{i-1}, \tag{2}$$

where x_i is the ith element of $\mathbf{x}$. This then allows operations such as vector-summations and dot products to be performed using binary (bit manipulation) operations. The dot-product, for example, can be represented as

$$\mathbf{a} \cdot \mathbf{b} = 32 - 2 \times \texttt{popcount}(\texttt{xor}(A, B)), \tag{3}$$

where both A and B are 32-bit unsigned integers holding the packed representations of the vectors $\mathbf{a}$, $\mathbf{b} \in \{-1, +1\}^{32}$. The operation 'popcount' (also known as Hamming weight calculation) is a function for computing the number of bits set to 1, which can essentially simulate vector summation. The operation `xor` in Eq. 3 is the bit-wise 'xor' operation.

C. Acceleration by Bit Manipulation Instructions

Looking at Eq. 3 we see that both 'xor' and 'popcount' are used in inference of binarized CNNs to perform an operation that emulates multiplication for packed weights; this means that both for fully-connected and convolutional layers 'xor'

and 'popcount' are in heavy use and offer a clear optimization target.

The hardware implementation of 'xor' can be found on any programmable processor, whereas a hardware implementation for 'popcount' is mostly available on graphics processing units or CPU SIMD extensions such as ARM NEON. For our target processor, the PULPino microcontroller, the base ISA does not include 'popcount' – this instruction is only present in the *bit manipulation extension* of RISC-V that is still under development [23].

In our experiments, in cases where the target processor did not have a hardware instruction for 'popcount', the LLVM C language description[1] shown in Algorithm 1 was called through __builtin_popcount().

Algorithm 1 LLVM 'popcount', i.e. Hamming weight

```
int32 popcountsi2 (int32 a) {
  uint32 x = (uint32) a;
  x = x − ((x >> 1) & 0x55555555);
  x = ((x >> 2) & 0x33333333) + (x & 0x33333333);
  x = (x + (x >> 4)) & 0x0F0F0F0F;
  x = (x + (x >> 16));
  return (x + (x >> 8)) & 0x0000003F;
}
```

V. EXPERIMENTS

The experimental evaluation of this work consisted of two parts: 1) evaluating the effect of the software-based binarization optimization for our image classifier, and 2) evaluating the effect of the 'popcount' custom instruction on the binarized classifier. Unfortunately, as our ultimate target platform was the PULPino microcontroller for IoT devices, it was not possible to benchmark the original *non-binarized* vehicle classifier on this device as it has no hardware support for floating point computations. Hence it was necessary to use two different target platforms to complete our experiments, and these platforms are summarized in Table II.

The ARM Cortex A53 core is a powerful mobile processor and in our experiments the processor was used under Linux for benchmarking a C language implementation of the original vehicle classifier [20], as well as for the C language implementation of the binarized vehicle classifier.

Experiments on the PULPino microcontroller platform were performed in a simulation environment, which is described next.

A. The ETISS Simulator

The RISC-V ISA is still in a phase of development, as for example the specification is not officially standardized yet. Still, the central components of the specification have matured and have been used to fabricate various chips such as the SiFive FE310 SoC [33]. The application being evaluated in this work however requires the *bit manipulation instruction* extension ('B extension') of the RISC-V ISA. This extension is still in active development [23] and not part of the current specification. Therefore, there is no RISC-V chip available that could be used for evaluating our results, however an alternative way to estimate the performance gain achievable through custom instructions is by simulation.

An RTL (Register-Transfer Level) hardware simulation would not be suitable for fast prototyping as the micro-architecture should be modified to enable the execution of the chosen custom instructions. Additionally, for a time-consuming workload such as our CNN application, the RTL simulation time would be prohibitively high.

The Extensible Translating Instruction Set Simulator (ETISS) focuses on extensibility [34] to support fast prototyping. As ETISS already supports the standard RISC-V base instruction sets, contains a virtual prototype of the PULPino [21] SoC, and allows profiling the application execution time, the use of this simulator was a natural decision our binarized image classifier application.

B. Implementation of the Popcount Instruction

As the PULPino virtual prototype of ETISS currently only supports the RISC-V base ISA, a temporary modification of the virtual prototype was required to enable profiling with support for 'popcount'. From ETISS execution traces it was discovered that the 'xori' instruction of the RISC-V base ISA remained almost unused throughout the whole execution of the binarized vehicle classifier. Therefore, in the PULPino virtual prototype the functional description of 'xori' was modified to provide alternative functionality, i.e. 'popcount', toggled by the value of the 2nd instruction operand.

In the software implementation of the binarized vehicle classifier, the calls to 'popcount' were then replaced with inline assembly calls to 'xori' with the specific operand value that would invoke 'popcount' behavior.

C. Execution Time and Memory Footprint Analysis

Table III shows the experimental results for both A53 and PULPino. From top to bottom the table rows report execution time on A53, execution time on PULPino, data memory footprint, PULPino instruction memory footprint, and CNN classification accuracy.

Looking at the A53 results it can be seen that binarization alone reduced the execution time by more than 80%, and dropped the data memory usage close to 95% when compared to the original floating point C version.

Acceleration by the hardware 'popcount' instruction reduced the computation time of the binarized vehicle classifier by around 55% on the PULPino platform, and also reduced the instruction memory footprint by around 2 kB. The reason for the 55% reduction in execution time can be seen from Table IV that shows the count of executed instructions on the PULPino platform for the binarized vehicle classifier with and without the hardware 'popcount' instruction: the code version that calls the hardware 'popcount' instruction has respectively

[1] https://github.com/sifive/riscv-llvm/blob/master/compiler-rt/lib/builtins/popcountsi2.c

TABLE II
PLATFORMS USED FOR EXPERIMENTS.

Tag	CPU	Platform type	Compiler	Operating System
A53	ARM Cortex A53 (1416 MHz)	Silicon SoC	g++ 5.4.0	Linux Firefly 4.4
PULPino	PULPino (33 MHz)	Virtual prototype on ETISS	riscv32-unknown-elf-gcc 7.1.1	*n/a*

TABLE III
EXECUTION TIME, MEMORY FOOTPRINT AND ACCURACY

Application version Arithmetic	Baseline float32	Binarized int32	Bin+pop int32
A53 Execution time	0.362 s	0.057 s	-
PULPino Exec. time	-	2.62 s	1.18 s
Data Memory	7.2 MB	369 kB	369 kB
Pulpino Instr. Memory	-	21 kB	19 kB
Accuracy [14]	97.09%	92.52%	92.52%

TABLE IV
NUMBER OF EXECUTED INSTRUCTIONS

Instruction name	Binarized int32	Bin+pop int32
lw	8797430	8797417
lbu	272	272
addi	6372539	6354083
slli	2801668	2801668
popcount/xori[2]	4	**3302052**
srli	**16510241**	1
srai	4	4
ori	1	1
andi	3302062	14
sb	268	268
sh	4	4
sw	782165	782109
add	**16704267**	3496013
mul	**0**	0
sub	**3670893**	368845
sll	18632	18632
slt	2553032	2553032
xor	3302048	3302048
or	2451656	2451656
and	**13208192**	0
bne	3232555	3232555
blt	0	0
bge	370058	370058
bltu	4	4
jalr	39	39
jal	57	57
csrrw	1	1
Total	***84078092***	***37830833***

55% less executed instructions. This is because if there is no hardware support for 'popcount', the functionality must be implemented by means of several regular instructions, which can be seen in increased execution counts of 'srli', 'and', 'sub' and 'add' instructions for the binarized version without the hardware 'popcount' instruction. Algorithm 1 shows that these instructions are needed for the software implementation of 'popcount'

The accuracy results shown in Table III are identical to our previous work on binarization that targeted graphics processing units [14].

VI. CONCLUSIONS

In this paper we have presented a convolutional neural network based vehicle image classifier that has been optimized for real-time execution and small memory footprint by a technique called *binarization*. We show that by using 'popcount', a custom instruction in our target processor, the runtime of the binarized image classifier can be reduced by 55%. This result is important due to the fact that 'popcount' has been proposed to be included to a standardized instruction set extension ('B extension') of the recently introduced open source RISC-V instruction set architecture. Besides RISC-V, 'popcount' is already supported in graphics processing units and e.g. in the NEON SIMD extension of ARM processors.

Our work shows that the software-based *binarization* transformation coupled with the hardware-based 'popcount' instruction yields an extremely powerful combination for optimizing inference of convolutional neural networks. Together, the memory footprint is reduced by close to 95%, and execution time is reduced by a magnitude while maintaining an acceptable loss in accuracy. As a results, image classification is performed in 1.18 seconds on the tiny 33 MHz RISC-V microcontroller that is well suited for IoT applications.

As binarization inevitably reduces classification accuracy (most clearly on larger datasets), a potential step for improving accuracy would be the adoption of *heterogeneous bitwidth binarization* [35]. This approach degrades accuracy considerably less than full binarization, already when on average 1.4 bits per weight are used [35].

[2] 'popcount' implemented as 'xori' alternative behavior

ACKNOWLEDGMENT

This work was partially funded by the Academy of Finland project 309903 CoEfNet, and by the ITEA3 project 16018 COMPACT (Business Finland diary number 3098/31/2017, German ministry of education and research reference number 01IS17028).

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 770–778.

[2] J. Dai, K. He, Y. Li, S. Ren, and J. Sun, "Instance-sensitive fully convolutional networks," in *European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 534–549.

[3] J. Johnson, A. Karpathy, and L. Fei-Fei, "DenseCap: Fully convolutional localization networks for dense captioning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4565–4574.

[4] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *IEEE conference on computer vision and pattern recognition (CVPR)*, 2017, pp. 7263–7271.

[5] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[6] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[7] M. Alioto and M. Shahghasemi, "The Internet of Things on its edge: Trends toward its tipping point," *IEEE Consumer Electronics Magazine*, vol. 7, no. 1, pp. 77–87, 2018.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–283.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016.

[10] J. Shen, Y. Huang, Z. Wang, Y. Qiao, M. Wen, and C. Zhang, "Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2018, pp. 97–106.

[11] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *International Workshop on Frontiers in Handwriting Recognition*, 2006.

[12] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 243–254.

[13] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," in *International Conference on Learning Representations (ICLR) Workshops*, 2018.

[14] M. Khan, H. Huttunen, and J. Boutellier, "Binarized convolutional neural networks for efficient inference on GPUs," in *European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 682–686.

[15] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *British Machine Vision Conference (BMVC)*. BMVA Press, 2014.

[16] 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 preprint arXiv:1704.04861*, 2017.

[17] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[18] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.

[19] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2017, pp. 65–74.

[20] H. Huttunen, F. S. Yancheshmeh, and K. Chen, "Car type recognition with deep neural networks," in *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2016, pp. 1115–1120.

[21] A. Traber, F. Zaruba, S. Stucki, A. Pullini, G. Haugou, E. Flamand, F. K. Gurkaynak, and L. Benini, "PULPino: A small single-core RISC-V SoC," in *RISC-V Workshop*, 2016.

[22] *The RISC-V Instruction Set Manual*, RISC-V Foundation, 2017, version 2.2.

[23] *RISC-V Bitmanip Extension*, Clifford Wolf, 2019, version 0.37.

[24] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 525–542.

[25] F. Pedersoli, G. Tzanetakis, and A. Tagliasacchi, "Espresso: Efficient forward propagation for binary deep neural networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[26] H. Park, D. Kim, J. Ahn, and S. Yoo, "Zero and data reuse-aware fast convolution for deep neural networks on GPU," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2016, pp. 1–10.

[27] F. Conti, P. D. Schiavone, and L. Benini, "XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, 2018.

[28] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *arXiv preprint arXiv:1801.01203*, 2018.

[29] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[30] J.-H. Hoepman and B. Jacobs, "Increased security through open source," *arXiv preprint arXiv:0801.3924*, 2008.

[31] B. Witten, C. Landwehr, and M. Caloyannides, "Does open source improve system security?" *IEEE Software*, vol. 18, no. 5, pp. 57–61, 2001.

[32] C. Cowan, "Software security for open-source systems," *IEEE Security & Privacy*, vol. 99, no. 1, pp. 38–45, 2003.

[33] *SiFive FE310-G000 Manual*, SiFive, Inc., 2017, version v2p3.

[34] D. Mueller-Gritschneder, M. Dittrich, M. Greim, K. Devarajegowda, W. Ecker, and U. Schlichtmann, "The extendable translating instruction set simulator (ETISS) interlinked with an MDA framework for fast RISC prototyping," in *International Symposium on Rapid System Prototyping (RSP)*. IEEE, 2017, pp. 79–84.

[35] J. Fromm, S. Patel, and M. Philipose, "Heterogeneous bitwidth binarization in convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 4006–4015.