Journal of Signal Processing Systems (2023) 95:1115–1126
https://doi.org/10.1007/s11265-023-01869-5

Accelerating image processing using reduced precision calculation convolution engines

Narayan Pokhrel[1] · Sakari Snäll[1] · Olli I. Heimo[1,2] · Uruj Sarwar[1] · Antti Airola[1] · Tero Säntti[1]

Received: 1 April 2021 / Revised: 13 March 2023 / Accepted: 11 April 2023 / Published online: 9 May 2023

Abstract
In this paper a method of accelerating image processing using convolution engines with reduced precision calculation is presented. The convolution engines are designed to be used with the Pulpissimo platform with RISC-V System-on-Chip. The aim is to move the calculation to the edge. The proposed linear convolution engines operate on 8-bit data set and the logarithmic convolution engine operates on 4-bit reduced precision data. The data reduction is done by using a logarithmic number space. Diminishing the size of the data to be processed reduces the amount of required memory, requirement for memory bandwidth, required computation, and required hardware area while simultaneously increasing the performance. This performance could benefit modern AI and image processing applications, especially in mobile and other battery-operated devices. The results show that the computation in the linear convolution engine is 91 times faster and computation in the logarithmic convolution engine is 122 times faster than in the RISC-V core with plain RISC-V instructions.

Keywords Convolution Engine · Image Processing · Reduced Precision Calculation · FPGA · System-on-Chip · Artificial Intelligence

1 Introduction

Technological development requires a constant increase in both speed and efficiency, combined with diminished costs, power consumption, and use of space. As physics gives us some limits, the mathematical models combined with embedded design can give us answers. The problem comes easiest in image processing where a human can easily identify a cat apart from a bear or a traffic light from a lorry, whereas a computer yet this day requires a lot of data to make such distinctions [1].

Yet, if the calculation of the simplest of analysis is done in the edge, this analysis might have an advantage against the human eye. As the eye of the beholder is indeed digital and turned towards the task ahead, the effectivity – as shown in this paper – can also be increased not only in benefit of the digital system, but also in benefit of the embedded design of the eye itself [2].

Machines equipped with a general-purpose processor (GPP) can handle a wide range of computations, ranging from simple embedded control to large-scale server workloads. The programming and running of an application in this kind of platform are relatively straightforward because of well-developed development tools and programming abstractions [3]. The general-purpose nature of a GPP makes them rather efficient for any task, but the specialized nature of a dedicated hardware accelerator allows focusing on only one task and thus increases the efficiency in that task. The execution of any kind of algorithms in GPP is far slower than the dedicated hardware accelerators. The hardware accelerators use a smaller area in the chip and can deliver high performance [4].

Most computer vision applications and real-time computation require complex computation by a processor. The GPP could not provide the computation needed for real-time processing. The convolution algorithm needs a relatively large amount of processing capacity, as each pixel is processed many times in several computation units to get the result. The proposed convolution engine is a dedicated hardware accelerator, which implements an image convolution

✉ Olli I. Heimo
olli.heimo@utu.fi

[1] Department of Computing, University of Turku, Turku, Finland

[2] Turku School of Economics, University of Turku, Turku, Finland

algorithm and delivers high performance in computer vision, videography, and photography. The proposed convolution engine reduces the memory read and write overhead by reutilizing pixel data from the previous computations. The computation for one image pixel by a 3×3 filter kernel needs a 3×3 image pixel window and arithmetic operations of 9 multiplications and 8 additions. The GPP needs many clock cycles to perform this computation, but a dedicated convolution engine can perform this operation in one clock cycle. A high-performance convolution engine contains many multiplication and addition units in parallel to increase throughput. The number of clock cycles for a GPP to complete the computation depends on the architecture of the processor and programming model. Typically, all of the separate multiplications and additions are executed sequentially in a GPP [4].

Processing images with convolution is hardly a new innovation. However, it is still a costly function when run with traditional methods of calculation and thus for the technology to evolve, a more efficient method should be found [5]. As the power consumption and calculation effectivity come into question – common cases in embedded systems – one is at crossroads. Whereas the standing solutions for different CPUs (Central Processing Units) offer different prizes (prize set at power consumption, heat production, and monetary prize), some graphics processing unit (GPU) additions give a flexible yet usually more monetarily increased prize yet being rather general purpose instead of dedicated accelerator sharing the problems of general-purpose units. Moreover, if the task is simplified enough, sufficient results can be implemented in various methods, for example, using only integer values instead of IEEE 754 [6] floating-point values. The switch to simpler data types can be done, if higher performance and/or reduced memory usage is required. Hence, reducing memory accesses to shared memory is an improvement, since memory operations are generally slow. Accessing memory fewer times also releases time for other components to use the memory bus [7].

Image processing is a digital operation widely used in various applications, for example pattern recognition or image classification. There are various ways to process digital images, such as using point operations, where a function is applied to each pixel, resulting in each pixel outputting the applied function [8].

On the other hand, group operations calculate the new pixel value from both, the original pixel and the neighboring pixels, making the template convolution a many-to-one function, where all the end pixels are independently calculated by the template kernel coefficients. Whereas, the human eye is rather keen on these kinds of transformations, computer calculation can suffer from efficiency issues when analyzing these kinds of issues. Moreover, a single bad value in imagining and thus in mathematical point operation could have significance in the computational analysis of the image, as the variations will skew the average differently from the median [9].

In the proposed engines, we have implemented a convolution engine to a RISC-V SoC in the Pulpissimo platform, with the focus on using reduced precision calculation to enhance the overall effectivity of the image processing. The convolution engine uses Direct Memory Access [10] reducing the CPU load. The chip is simulated, prototyped using FPGA and is ready for the tape-out process, and this article is being written during this course of time. The main focus and application domain intended for these engines is machine learning, and even more specifically performing it in an edge device. This means that the main objective is minimizing energy and resource usage while maintaining structural information in the images. Visual quality, as perceived by a human, is not a major factor. The long-term target is creating a full set of arithmetic units capable of using the proposed reduced precision style, allowing full systems to use the same approach consistently throughout the pipeline.

2 Convolution engines

2.1 Convolution in image processing

Image processing applications have been constantly advancing, with the advantage of applying operations on image-based data to extract detailed information. This also involves using several image acquisition methods for image analysis, which could differ from application to application. One of the main methods of image processing can be regarded as convolution, which is simply a technique of image transformation by a kernel covering each of the pixel values. This kernel itself is a matrix that is responsible for performing convolution in the first place, and it further indicates what type of image transformation should occur. As many advantages can be observed in image processing, it also comprises challenges when it comes to increased workloads. This is why the general-purpose hardware might not be properly optimized for high data parallelism. In such circumstances, Single Instruction Multiple Data (SIMD) units and GPUs are used, but these do require more power. Accelerators based on Application-Specific Integrated Circuits (ASICs) are also being developed that are power efficient [11].

2.2 Convolution engines and GPP

The use of GPP has been diversifying immensely; however, with this benefit of constantly being advanced and flexible, they do utilize a higher price, because 99% of the energy is being wasted, per overhead, in programmability [11]. The

research study performed by Qadeer et al. [11] indicated that the defined energy waste could be decreased by altering data storage and designing structures, along with their relation to data locality and dataflow in algorithmic approaches. This is why instead of aiming entirely on programmability, major data flow patterns could be focused in a domain. With such methods, efficient engines can be designed and used in different applications with respect to the execution domain.

These engines can be further programmed based on the application. For this purpose, the mentioned study [11] worked on a Convolution Engine and identified it as a programmable processor. The engine is used for convolution-based data flow in the applications of video processing, computer vision, and computational photography. Convolution engines contribute to energy efficiency by performing several operations on each memory access, reducing data transmission overheads, and utilizing data-reuse patterns. Qadeer et al. [11] established their engine that was targeted to a factor of 2–3 × of the energy, and also the area efficiency regarding customized units based on a single kernel. The study further elaborated that by using the convolution engine, area efficiency, and energy were enhanced by 8–15 × in comparison to SIMD-based engines.

2.3 Convolution engines in machine learning

In machine learning, convolutional neural networks (CNN) [12] have become the dominant approach to image analysis. The convolution kernels allow modeling hierarchical data representations, where units in the first layers extract simple shapes such as edges or corners, and subsequent layers combine these into more high-level abstract patterns. The weights of the kernels as well as other network parameters are automatically learned from data typically via stochastic gradient descent optimization. Advances in training deep CNN architectures have resulted in state-of-the-art and even human-level performance in tasks such as object detection and segmentation, image classification, video processing, speech recognition, and natural language processing [13]. Yet the cost of these advances has been a steep increase in computational requirements both for training and deploying the networks, motivating the need for solutions to improve efficiency such as specialized GPU, or increasingly also FPGA and other ASIC-based hardware accelerators [14].

The study performed by Liu et al. [15] presents an FPGA-based CNN accelerator that provides reduced power consumption, in comparison to GPUs and ASICs. Despite the advantage of less power consumption, there are still challenges concerning complex and large computational features of CNN and the insufficient resources of FPGA. These challenges are encountered when designing such hardware accelerators. The mentioned study [15] demonstrated a CNN accelerator that enhances the standardized convolution and also the depth-wise separable convolution. A CNN consists of several layers for processing the input image data. A specific convolutional layer is also present that is mainly required to apply convolution calculation on the input data by using convolution kernels. The designing of convolution engines is based on an architecture that could contribute to good performance, efficient power, and the ease of having multiple convolution-based algorithms as described in CNN. This could support handling workloads and enhancing efficiency in image processing. Another research study also regarding CNN [16] indicated that reduced precision data can be utilized within the network layers, as it helps in decreasing energy and supports better performance. This further provides the platform for having larger networks in CNN-based applications. Due to such advantages, another study [17] developed a CNN accelerator for reducing computation. This accelerator used the method of precision-cascading (PC).

Beyond standard CNN architectures, convolution acceleration could have also a significant impact on the efficiency of a wide range of other machine-learning approaches that employ convolutions. These include methods such as convolutional kernel networks based on the theory of reproducing kernels [18] probabilistic methods such as deep convolutional Gaussian processes [19], as well as various extensions of CNNs to unsupervised [20] or reinforcement [21] learning. Since a given application typically has several layers of CNN, any benefits in the execution are multiplied by the number of convolution runs. This applies to savings in power and execution time, and can be further increased by pipelining accelerators like the ones presented here. This, however, does fall outside the scope of this paper.

3 Reducing precision

Approximate computing or *reduced precision calculation* is a technique producing deliberately inaccurate results compared to more precise algorithms. As precision is not always the ultimate key as different solutions require different thresholds, some precision can be sacrificed to gain more efficiency. For example, every vote counts when calculating votes from a national election, but when conducting opinion polls before the actual election, a good estimate is enough.

One technique to gain efficiency from the reduced precision calculation is reduced memory usage. As the values being processed require less space, they also produce less traffic – in addition to other effects such as reduced calculation time. Another technique is to transform the data into a different mathematical representation, for example convert it to logarithmic scale, as presented by Miyashita et al. [22].

An 8-bit integer input can produce a logarithmic output that can fit into 3 bits. When converted back to linear

space, it can only represent values ranging from 0 to 128. However, the output requires 4 bits if the rounding method is used. Yet the 4-bit value can represent the 8-bit input in full range. In a mathematical sense, the logarithmic space has clear advantages, as the linear space multiplications are transformed into additions, as shown in the following equation:

$$log(x1 * x2... * xn) = log(x1) + log(x2)... + log(xn)$$

In hardware, addition requires fewer gates than multiplication enabling a marginal size reduction in the overall design.

In recent years, many teams have evaluated the suitability of using logarithmic number precision in accordance with convolutional neural networks [22-24]. The results indicate that logarithmic representation not only saves the memory storage and data movement requirements, but is also able to preserve the recognition accuracy of the CNN as compared to more conventional number representations.

Using the logarithm numbers in convolutions, also the arithmetic operations are transformed to more efficient realizations, where e.g., the bulky digital multiplication can be avoided.

4 Design

The proposed convolution engine is implemented into Pulpissimo as a hardware accelerator. Pulpissimo, an open-source microcontroller architecture, is developed by ETH Zurich and the University of Bologna [25]. It allows the development of customized platforms with a reference implementation for both single- and multi-core SoC illustrated in Fig. 1. The HWPE (Hardware Processing Engine) contains the integrated convolution engine whereas the peripheral bus transfers data and interrupts. The engine is also able to notify the CPU about completed processes and error messages. Data transfers are done with TCDMI (Tightly Coupled Data Memory Interconnect) and to accomplish data transferring, the convolution engine is connected to TCDMI with AXI (Advanced eXtensible Interface) bus.

The Pulpissimo platform implements RISC-V, an open-source ISA, whose specifications are maintained and ratified by RISC-V international contributing members [26]. The test programs were made directly with the RISC-V assembly with C and C++ support and with GCC [27]. The convolution engine was designed to be modular and reusable. Figure 2 illustrates the architecture of the hardware convolution engine consisting of kernel and linear buffers, in/out fifos, and multiplication accumulation unit. Every modular unit contains ready and valid signals known as handshake signals

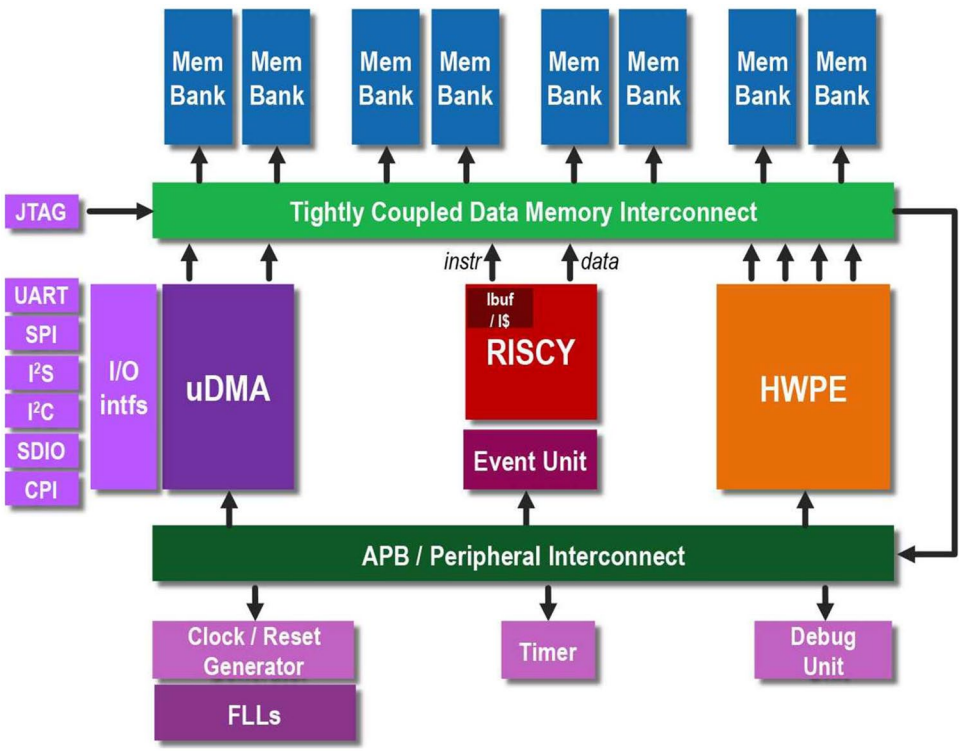

Fig. 1 Overview of Pulpissimo [25]

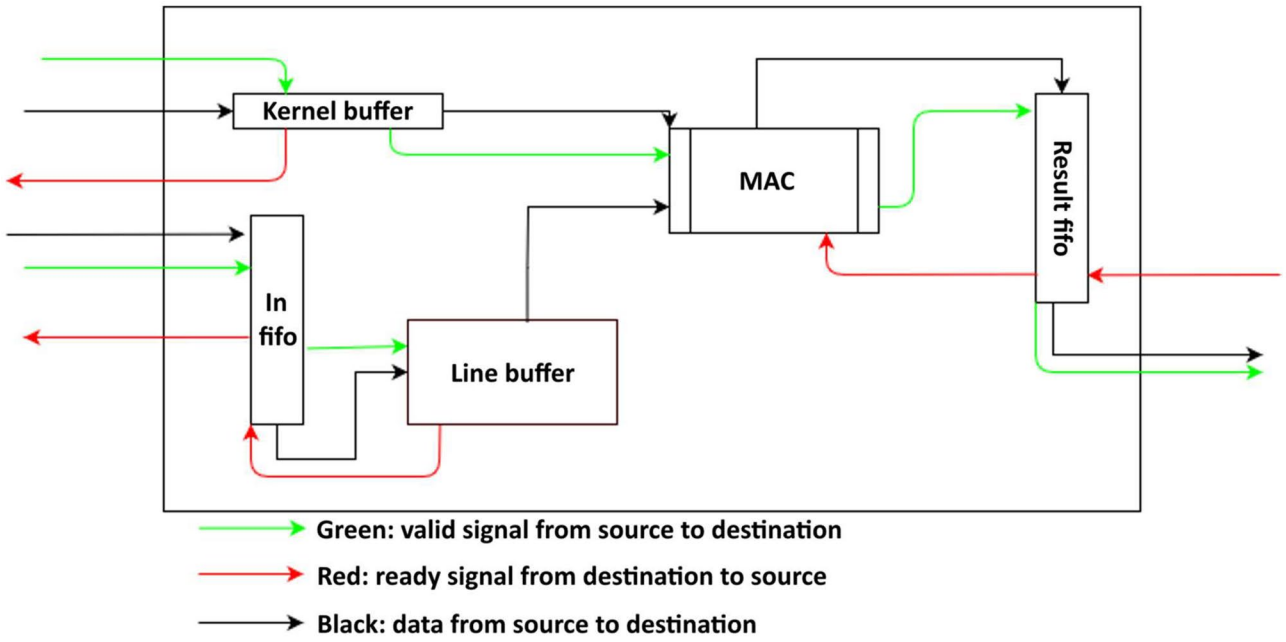

Fig. 2 Engine data flow

to communicate when the data is ready to be transferred, shown as red and green arrows [28].

The communication between the CPU via the AX4-lite bus and data communication with the memory cluster via the AXI4 bus are provided by the top-level wrapper. AXI4-lite bus also delivers the CPU convolution configurations. The configuration parameters are image data with filter kernel coefficients, Image dimensions, memory address, result image saving starting memory address, AXI burst size, and engine start signal. Two separate software resets for the engine and kernel buffer are used, making running several convolutions with the same kernel coefficient possible.

The convolution engine receives the image data directly from the memory (Direct Memory Access, DMA). The image starting address and output address are sent from the CPU to the engine, which handles the data reading and writing. As the CPU has sent all needed data to the engine, it can send a start signal for the processing to commence.

As both these proposed engines need to send and receive data from memory and CPU, there is a requirement for control over how buses are connected. The engines must be able to both read and write from memory. In AXI this means the engines are masters and the memory is a slave. The user must be able to read the engine status and send control data to it. Therefore, in the AXI-bus the CPU is the master and the engines are slaves. The crossbar component manages the existing connections for the engines, as illustrated in Fig. 3. The AXI-bus from the CPU has a 32-bit wide data channel with a 64-bit wide memory channel. The engines receive commands from the CPU with AXI4-Lite. The 64-bit wide bus to memory can utilize burst modes from AXI4.

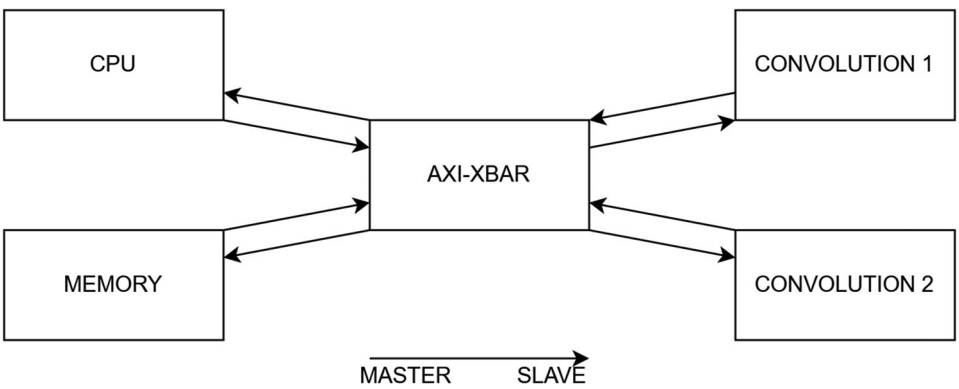

Fig. 3 Data paths from convolution engines

5 Results

To measure the performance of the convolution engines test cases were done. The tests were executed on a software simulator and on an FPGA. RT-Profiling tool was used to measure how many cycles the whole process used.

Roughly 60–90 cycles are used to send configuration values to the engine and to read the status register when the process is done. Processing 16×16 image in the logarithmic engine takes only 197 cycles which is 182 times less than a full software implementation.

An image in the logarithmic domain requires 50% less memory than an image in the linear domain. That is the main reason why a logarithmic engine uses fewer cycles compared to a linear engine; it needs to move fewer data around. Converting an image to the logarithmic domain and back to the linear domain is done in the CPU. 16×16 image could be processed 8 times in the linear, in the time that it takes from the CPU to do the data conversion both ways. This means that the logarithmic engine is more suitable in use cases where the data can be kept in the logarithmic domain or the memory available is heavily constrained (see Figs. 4, 5).

The total cell count of the logarithmic engine is 70% of the linear engine. Both of the engines also have 3 row buffers which are separated from the engine and not counted in the total cell count of the engine. Those row buffers can be half the size for a logarithmic engine. Both convolution engines have four AUs (Arithmetic Unit). A logarithmic AU is 13% of the size of a linear AU. If the chip were to have multiple convolution engines, adding new logarithmic engines would scale better size-wise.

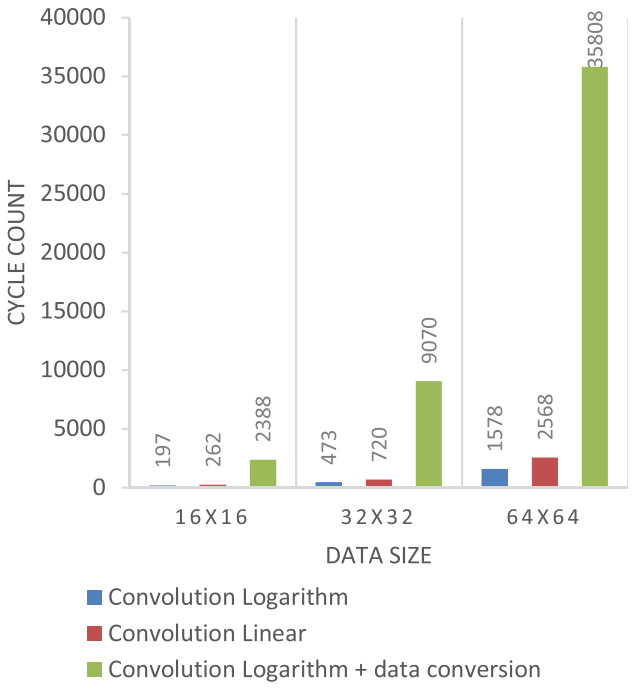

Fig. 4 Cycle counts for various image sizes using linear and logarithmic convolution engines including data conversion

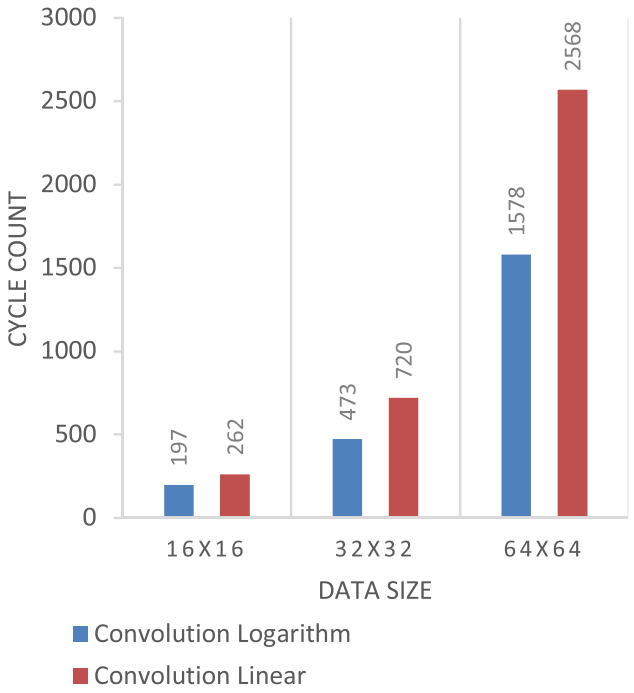

Fig. 5 Cycle counts for various image sizes using linear and logarithmic convolution engines

To test the performance and instruction execution, 4 different image data sizes (8×8, 16×16, 32×32, and 64×64) are used. For every image data size, 3 different program execution methods are used. The methods and process are explained as follows:

A. RI5CY only:
   The test is run in the RISC-V core only using standard RISC-V only and standard RISC-V and pulp-specific instructions.

1. Compiling test using standard RISC-V instructions:
   In this method, the test is compiled using RV32IMC, which contains the Base integer instructions and, multiplication/division and compressed extended instructions. In this process, instruction size and clock cycles are significantly higher than in other methods.
2. Compiling test using Standard RISC-V instructions and Pulp specific ISA extensions:
   In this method, the test is compiled using RV32IMC and xpulpv2 (pulp specific ISA extensions). The clock cycle and instruction size are lower than the standard RISC-V.

B. Convolution Engine:

The test is run in hardware convolution engine. The speed of execution does not matter if it complied with RISC-V only or RISC-V and Pulp specific instructions. The number of clock cycles, and instructions executed are lower than the software run methods. In the case of software computation, the dot product unit (pulp extended) makes the operation significantly faster than the standard RISC-V computation. Even though the PULP compiler itself can produce better results, it cannot utilize all instructions efficiently. The optimized c code is written using the GCC build-ins for RISC-V ISA extensions, which calls pulp specific extended instructions directly and runs on extended architecture. RI5CY has the dot product unit which supports 8 bits and 16 bits vector operands [29]. This dot product unit produces the result of 4 multiplications and 3 additions in a single operation. The use of this unit significantly reduces the number of clock cycles and instruction executions in the image convolution algorithm.

Figures 6, 7, 8, 9 illustrate the number of clock cycles and instructions executed in Questasim simulations and FPGA test runs. The performance comparison graph shows the performance of the RI5CY core with pulp extensions and hardware convolution engine.

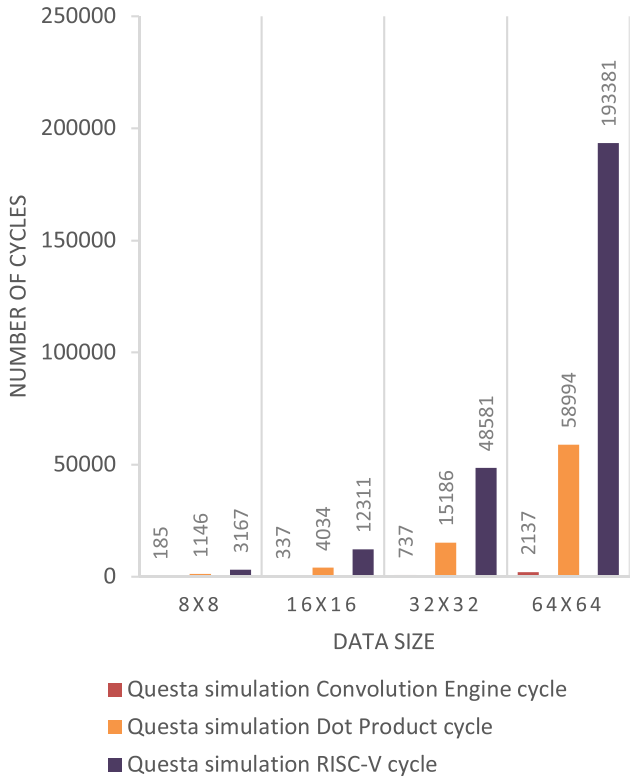

Fig. 6 Questa simulation result in terms of cycles

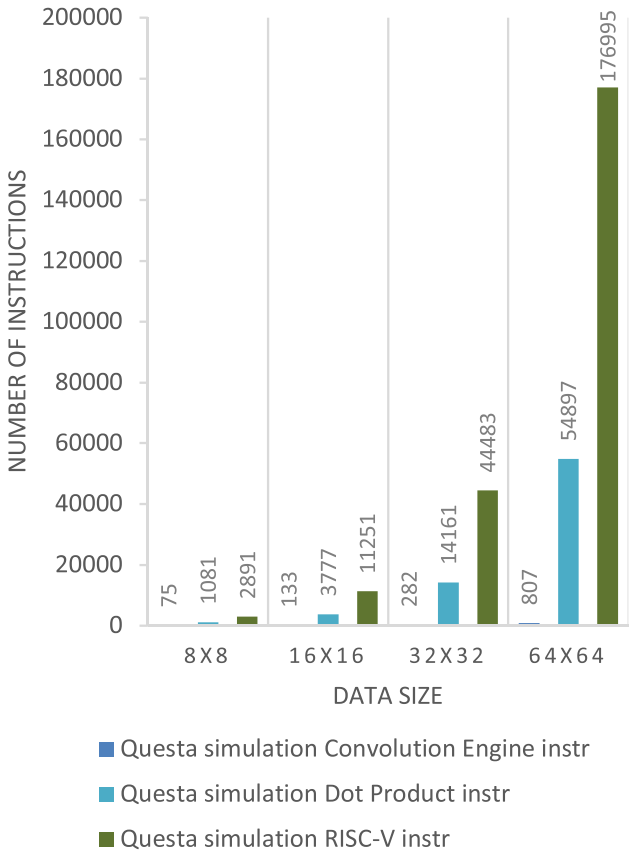

Fig. 7 Questa simulation result in terms of instructions

The number of clock cycles used when running the test in the convolution engine is very low. The engine is capable of fetching image pixels from L2 memory and streaming the result back to L2 memory with the start command by the RISC-V core after configurations. The hardware convolution engine runs 17 times faster than the RISC-V standard instructions and enabling the pulp specific instruction reduces the performance from 17 to 6 times on image data size 8×8. Likewise, the performance gain by the convolution engine is 91 times than standard RISC-V and 26 times when enabling pulp specific instructions on image data size 64×64. The performance gain increases with image data size. This is due to initial configurations needed for the engine, and full utilization of transferred pixels by saving in line buffer for future image windows and in shift buffer for the next image window. But in the case of RI5CY, the pixels are moving from memory to the core register more than one time.

In the case of software run, the computation is very complex, instruction fetch, decode, data load, execution, result write back, all these operations are happening all the time in pipeline methods. So, each result pixel is produced from the number of multiplication and addition operations. From the above results, using dot product unit significantly reduces

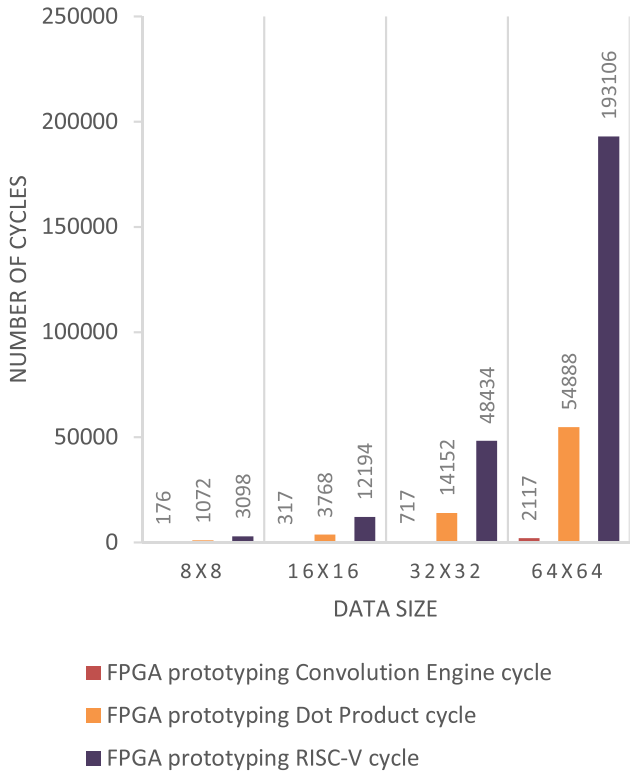

Fig. 8 FPGA prototyping result in terms of cycles

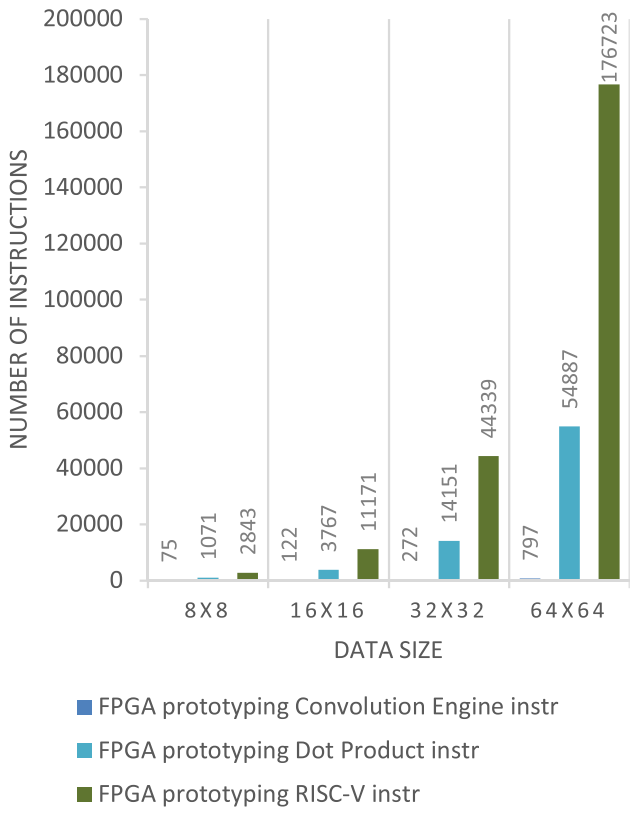

Fig. 9 FPGA prototyping result in terms of instructions

the active clock cycles. The image convolution algorithm runs 3 times faster in the RISCY core using pulp specific extended instructions. Similarly, increasing the size of image data increases the performance difference between the RISC-V standard and RISC-V pulp extensions. The slight variation between the number of clock cycles and the number of instructions executed is because of load data hazards and cycles waiting for instruction fetch. In the case of FPGA testing, load data hazard, cycles waiting for instruction fetch is zero when running convolution using RI5CY core only and negligible amount of load data hazards when using pulp extensions.

The overall result summarises that enabling pulp specific instruction reduces the program execution time and reduces the data load from memory. In the case of 3×3 convolution, it is possible to utilize the 6 pixels from the last 3×3 image window and load new 3 pixels. The hardware convolution engine runs even faster than pulp specific instructions, by fully utilizing the transferred pixels by storing them in line buffers. The hardware convolution engine requires less memory bandwidth and fewer clock cycles compared to the RISC-V core.

6 Quality analysis

Following the results, examining the trade-off needed to achieve all the benefits is prudent. In this case, the reduced memory footprint and increased performance are compensated by some loss of visual quality. Mainly this stems from using fewer bits per pixel. This effect was analyzed with a set of random images, all featuring outdoor scenes. The images were first scaled to similar size, so that image size would not distort the average results. Larger images would have more samples; thus they would have more weight in the analysis. The process then converted the images to normal grayscale. This was considered the ground truth. The comparison was done by converting all images to logarithmic scale, then directly back to grayscale. After that, the sum of absolute differences in the intensity values of each pixel was calculated. This sum is presented in the "Sum diff" column in Table 1. The difference was also divided for all the pixels, showing the average error in each pixel. This value is presented in unit [255.0.0], corresponding to normal 8-bit grayscale values. Finally, the error is shown also in percentage.

Considering the intended use case, further analysis was done to assess the impact for machine learning. Here the assumption was that the structures presented in the images would remain better intact than visual quality. To show that in practice, two different analysis sequences were run on the same images. First, a simple edge detection convolution, with coefficients 8 in the middle and -1 on all of the neighbors. This kernel was applied to both the normal

Table 1 Brightness distortion analysis

Image	X	Y	Sum diff	Per pixel	Relative error
1	320	213	516,801	7,58	2,97%
2	320	213	347,107	5,09	2,00%
3	320	212	791,387	11,67	4,57%
4	320	180	263,241	4,57	1,79%
5	320	207	500,387	7,55	2,96%
6	320	213	821,434	12,05	4,73%
7	320	214	730,949	10,67	4,19%
8	320	196	262,699	4,19	1,64%
9	320	214	641,341	9,37	3,67%
10	320	180	72,018	1,25	0,49%
11	320	213	980,831	14,39	5,64%
12	320	212	571,580	8,43	3,30%
Total			**6,499,775**	**8.23**	**3.22%**

grayscale image and the logarithmic version. The results are shown in Fig. 10. The left-hand side is completely in the normal binary grayscale domain, while the right-hand side is processed in the logarithmic domain, and converted back to binary for visualization after the processing. The results show some differences, mostly in the way the color gradients in the sky are handled, but most of the structures are strikingly similar.

To show, that other processing is also viable in the logarithmic domain, another test sequence was run. This time the convolution was used to perform gaussian blur on the subject image. A typical 3×3 kernel was used, with weights 1 for all corners, 2 for the sides, and 4 for the middle. Finally, the result was scaled back to the normal range by dividing by 16. This image was then used to find edges, by subtracting the original from this image. Again, the results are presented in Fig. 11 in the same order, all left-hand side frames are in the normal binary grayscale domain, and the right-hand side is processed in the logarithmic domain. This time there is more difference between the resulting edges. Again, most of the differences are in the sky, which had relatively smooth transitions in the original image. However, the actual structure is shown identified well, and some might even say the logarithmic version has less noise in the sky. Fine-tuning of thresholds could bring the results closer together, but these are untuned, for the sake of comparison of the numerical spaces, not optimized for visual quality or similarity. In Fig. 11, the first row shows the original images, then the next row shows the results of gaussian smoothing and finally, the last row shows the edges detected by subtracting the original from the smoothed image.

After looking at the results of actual operations, it can be seen that even though there is some degradation in terms of actual intensity values, it does not translate to major differences in the processing results. As a result, the techniques are suitable for the application domain of machine learning, but maybe not be recommended for domains where visual quality is of paramount importance. The sample images were chosen so that they had more intensity errors than average. This was done to avoid unwarranted optimism in the results. The test images were acquired from Pixabay, a site that offers fully free images.

Fig. 10 Results of edge detection by convolution (image number 7 in Table 1)

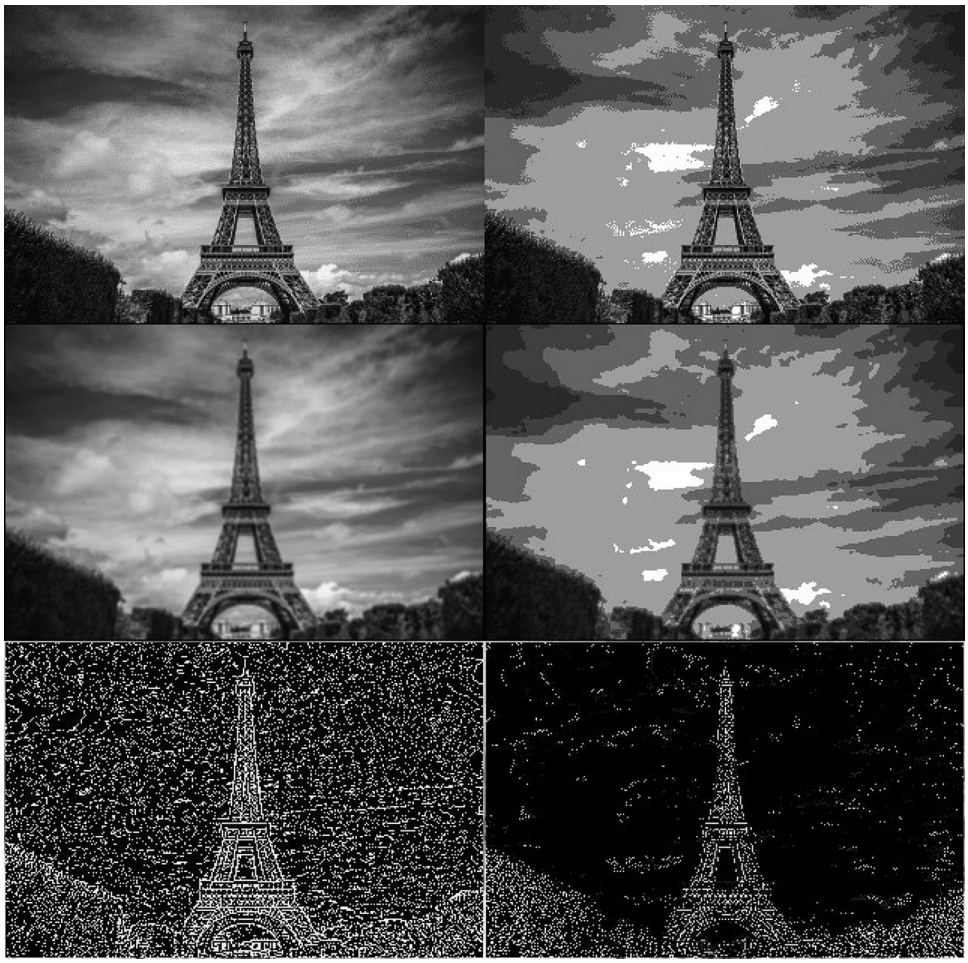

Fig. 11 Convolution applied for gaussian smoothing and the edge detection (image number 2 in Table 1)

7 Conclusions

The hardware accelerator implementation with reduced precision technique to process images is efficient and effective. The proposed engine gives a good indication of the possibilities of using logarithmic data instead of linear data. The images produced from the logarithmic engine have a visible quality degradation, but the structure compared to the precise method was similar which is in line with the results of Miyashita et al. [22]. More research on how useful the reduced precision images could be is still required.

The convolution engines proposed in this paper could be optimized further. As an example, increasing the number of AUs in the engine should increase the engine performance with an increased data transfer rate. Also, parallel convolution engines with DMAs to their own memories and crossbars should increase effectivity. These research questions, however, are left for future study.

However, even the current solution provides results that clearly show an increase in performance compared to a linear engine, by reducing 34% of all required cycles with 30% reduction in cell count. The computation in the linear convolution engine is 91 times faster and computation in the logarithmic convolution engine is 122 times faster than in the RISC-V core with plain RISC-V instructions. Whereas the data conversion in CPU requires irrefutably more, roughly 100 times more (see Fig. 5), cycles thus making the linear engine a preferable choice, but with the hardware accelerator the results justify and promotes the logarithmic engine. The best advantage is obtained, if all of the parts in an image processing pipeline use the same number format, thus avoiding conversions.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11265-023-01869-5.

Acknowledgements This work is part of the FitOptiVis project [30] funded by the ECSEL Joint Undertaking under grant number H2020-ECSEL-2017-2-783162. The authors thank Ari Paasio, Jenni Heimo, Lauri Koskinen, and Jani Silvander for their assistance. Additionally, Pixabay (https://pixabay.com/) is gratefully acknowledged for providing the test images used in the quality analysis.

Funding Open Access funding provided by University of Turku (UTU) including Turku University Central Hospital. This work is part of the FitOptiVis project [30] funded by the ECSEL Joint Undertaking under grant number H2020-ECSEL-2017–2-783162.

Data Availability Two M.Sc. (tech) theses about the subject will be published on the University of Turku website.

Code Availability Not applicable.

Declarations

Conflicts of interest/Competing interests Not applicable.

References

1. Chitradevi, B., & Srimathi, P. (2014). An overview on image processing techniques. *International Journal of Innovative Research in Computer and Communication Engineering*, (2(11), pp.6466–6472).
2. Viswanathan, V., & Hussein, R. (2017). Applications of image processing and real-time embedded systems in autonomous cars: a short review. *International Journal of Image Processing (IJIP),* (11(2), pp. 35).
3. Cardoso, J. M., Carvalho, T., Coutinho, J. G., Luk, W., Nobre, R., Diniz, P., & Petrov, Z. (2012). LARA: an aspect-oriented programming language for embedded systems. In *Proceedings of the 11th annual international conference on Aspect-oriented Software Development* (pp. 179–190). https://doi.org/10.1145/2162049.2162071
4. Qadeer, W., Hameed, R., Shacham, O., Venkatesan, P., Kozyrakis, C., & Horowitz, M. A. (2013). Convolution engine: balancing efficiency & flexibility in specialized computing. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (pp. 24–35). https://doi.org/10.1145/2485922.2485925
5. Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., Gholaminejad, A., Gonzalez, J., & Keutzer, K. (2018). Shift: A zero flop, zero parameter alternative to spatial convolutions. In Proceedings of the IEEE Conference on *Computer Vision and Pattern Recognition* (pp. 9127–9135).
6. IEEE (2019). "IEEE standard for floating-point arithmetic", *IEEE Std 754–2019 (Revision of IEEE 754–2008),* pp. 1–84, 2019. https://doi.org/10.1109/IEEESTD.2019.8766229
7. Khwa, W.S., Chen, J.J., Li, J.F., Si, X., Yang, E.Y., Sun, X., Liu, R., Chen, P.Y., Li, Q., Yu, S. & Chang, M.F., 2018, February. A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)* (pp. 496–498). https://doi.org/10.1109/ISSCC.2018.8310401
8. Wiercioch-Kuzianik, K., & Bąbel, P. (2019). Color hurts. The effect of color on pain perception. *Pain Medicine,* (20(10), pp. 1955–1962). https://doi.org/10.1093/pm/pny285
9. Nixon, M., & Aguado, A. (2019). *Feature extraction and image processing for computer vision*. Academic press.
10. Ma, Y., Suda, N., Cao, Y., Seo, J. S., & Vrudhula, S. (2016). Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)* (pp. 1–8). IEEE.
11. Qadeer, W., Hameed, R., Shacham, O., Venkatesan, P., Kozyrakis, C., & Horowitz, M. A. (2015). Convolution engine: balancing efficiency and flexibility in specialized computing. In *Communications of the ACM* (pp. 85–93). https://doi.org/10.1145/2735841
12. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation,* (1(4), pp. 541–551). https://doi.org/10.1162/neco.1989.1.4.541
13. Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review,* (53(8), pp. 5455–5516). https://doi.org/10.1007/s10462-020-09825-6
14. Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. (2020). The computational limits of deep learning. arXiv preprint arXiv:2007.05558.
15. Liu, B., Zou, D., Feng, L., Feng, S., Fu, P., & Li, J. (2019). An fpga-based cnn accelerator integrating depthwise separable convolution. *Electronics,* (8(3), pp. 281). https://doi.org/10.3390/electronics8030281
16. Judd, P., Albericio, J., Hetherington, T., Aamodt, T., Jerger, N. E., Urtasun, R., & Moshovos, A. (2015). Reduced-precision strategies for bounded memory in deep neural nets. arXiv preprint arXiv:1511.05236.
17. Kim, M. (2019). *Energy-Efficient ASIC Accelerators for Machine/Deep Learning Algorithms* (Doctoral dissertation, Arizona State University).
18. Mairal, J. (2016). End-to-end kernel learning with supervised convolutional kernel networks. In *30th Conference on Neural Information Processing Systems (NIPS 2016),* (pp. 1–16).
19. Blomqvist, K., Kaski, S., & Heinonen, M. (2019). Deep convolutional Gaussian processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 582–597). Springer, Cham. https://doi.org/10.1007/978-3-030-46147-8_35
20. Dosovitskiy, A., Fischer, P., Springenberg, J. T., Riedmiller, M., & Brox, T. (2015). Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence,* (38(9), pp. 1734–1747). https://doi.org/10.1109/TPAMI.2015.2496141
21. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *nature,* (518(7540), pp. 529–533). https://doi.org/10.1038/nature14236
22. Miyashita, D., Lee, E. H., & Murmann, B. (2016). Convolutional neural networks using logarithmic data representation. arXiv preprint arXiv:1603.01025.
23. Lee, E. H., Miyashita, D., Chai, E., Murmann, B., & Wong, S. S. (2017). Lognet: Energy-efficient neural networks using logarithmic computation. In 2017 *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 5900–5904). IEEE. https://doi.org/10.1109/ICASSP.2017.7953288
24. Vogel, S., Liang, M., Guntoro, A., Stechele, W., & Ascheid, G. (2018). Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base. In *Proceedings of the International Conference on Computer-Aided Design* (pp. 1–8). https://doi.org/10.1145/3240765.3240803
25. GitHub, Inc. (2021). PULPissimo. Retrieved January 13, 2021, from https://github.com/pulp-platform/pulpissimo

26. Waterman, A., Lee, Y., Patterson, D. A., & Asanovi, K. (2014). The risc-v instruction set manual. volume 1: User-level isa, version 2.0. California Univ Berkeley Dept of Electrical Engineering and Computer Sciences.
27. Stallman, R. M. (1988). Using the GNU Compiler Collection. *For GCC version,* (4(2)). Retrieved January 13, 2021, from https://gcc.gnu.org/onlinedocs/gcc.pdf
28. Fletcher C, W. (2009). Interfaces: Fifo (a.k.a. ready/valid). Retrieved March 3, 2021, from https://inst.eecs.berkeley.edu/~cs150/Documents/Interfaces.pdf
29. Gautschi, M., Schiavone, P. D., Traber, A., Loi, I., Pullini, A., Rossi, D., Flamand, E., Gürkaynak, F. K., & Benini, L. (2017). Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* (25(10), pp. 2700–2713). https://doi.org/10.1109/TVLSI.2017.2654506
30. The FitOptiVis ECSEL project: highly efficient distributed embedded image/video processing in cyber-physical systems, ACM Int'l Conf. on Computing Frontiers, 2019, pp. 333–338, https://doi.org/10.1145/3310273.3323437