ICT Express

Design Approach for Power and Resources Efficient CNN Accelerators

--Manuscript Draft--

Manuscript Number:	ICTE-D-21-00020
Article Type:	Research paper
Section/Category:	SI on Mobile Edge
Keywords:	CNN; Accelerator; Edge Computing; FPGA; Vessel Detection
Abstract:	Machine Learning tasks depend on neural network accelerators to achieve real-time execution. Aiming at improving the accelerators regarding performance, power and resources for CNNs of limited feature space the current study presents a design approach for architectures utilizing the logic and memory resources of a single FPGA targeting edge computing. The example accelerator for Vessel Detection operates on RGB images of size 8080 or sliding windows, it is trained for the “Ships in Satellite Imagery” and on a Xilinx VC707 completes the inference in .687 ms operating at 270 MHz and with 5 watts validates the design approach.

Design Approach for Power and Performance Efficient CNN Accelerators

Angelos Kyriakos, Elissaios-Alexios Papatheofanous, Charalampos Bezaitis, Dionysios Reisis*

Electronics lab, Physics Dpt, National and Kapodistrian University of Athens, Greece

Abstract

Machine Learning tasks depend on neural network accelerators to achieve real-time execution. Aiming at improving the accelerators regarding performance, power and resources for CNNs of limited feature space the current study presents a design approach for architectures utilizing the logic and memory resources of a single FPGA targeting edge computing. The example accelerator for Vessel Detection operates on RGB images of size 80×80 or sliding windows, it is trained for the "Ships in Satellite Imagery" and on a Xilinx VC707 completes the inference in .687 ms operating at 270 MHz and with 5 watts validates the design approach.

Key words: CNN, Accelerator, Edge Computing, FPGA, Vessel Detection

1. Introduction

Deep learning techniques based on trained deep convolutional neural networks (CNNs) impose the use of accelerators to accomplish real-time performance. The accelerators targeting edge and mobile computing have to be performance, power and resources efficient and the current paper presents a design approach for real-time classification FPGA accelerators that can be implemented with the logic and memory resources of a single FPGA device. The proposed approach focuses on CNN applications with relatively low feature space such as the classification problems that share similar characteristics between classes [1], [2] and CNNs requiring few convolution layers such as SAT-4/SAT-6 [3]. The proposed approach follows three phases. The first designs the application with the TensorFlow and the second transforms the model into a fixed-point bit accurate model (BAM) simulating the hardware calculations. For the third phase, we developed a library of algorithm specific blocks in VHDL implementing the CNN functions with fixed-point arithmetic. The third phase exploits these blocks to produce an accelerator with improved throughput and power consumption and also limits the development time.

We illustrate the approach by designing an FPGA accelerator that detects if there is a vessel [2], in the input image. The CNN is trained for the Planet's "Ships in Satellite Imagery" dataset [4] and on a Xilinx VC707 achieves almost 98% prediction accuracy and high throughput by classifying a 80 × 80 RGB 24 bits/pixel image in 0.68 msec. The accelerator can be used in a sliding window application for scenes up to 4K. To compare the FPGA's performance we execute our code on the low power Intel's Myriad2 processor [5] and the edge-computing NVIDIA's Jetson Nano [6] either on the Jetson's ARM processor or the GPU.

Related results for CNN FPGA accelerators are based mainly on the automated software development tools like the HLS [7], [8], [9]. Regarding the Vessel Detection presented as a case application, the widely known results focus on algorithmic techniques Faster-RCNN [10] and Single Shot MultiBox Detector (SSD) [11].

2. CNN Design Approach

The current section introduces the three phases of the proposed approach, the VHDL blocks and the mapping of the CNN onto the FPGA by using these blocks.

2.1. CNN Design Space Exploration

The first phase uses the TensorFlow estimator API to design the CNN's model targeting to fit within a single FPGA's resources. It focuses on the following factors: first, the neural networks for the above tasks can achieve

*Corresponding author
Email addresses: akyriakos@phys.uoa.gr (Angelos Kyriakos), eapapatheo@phys.uoa.gr (Elissaios-Alexios Papatheofanous), bezaitisc@phys.uoa.gr (Charalampos Bezaitis), dreisis@phys.uoa.gr (Dionysios Reisis)

a high accuracy rating even with a relatively small number of Convolution Layers [3]. Second, for relatively small images, the recognized objects tend to occupy a large portion of the image and hence, large and medium size convolution kernels suffice. Third, increasing the number of convolutional layers improves the CNN's accuracy, increases also the computational cost but reduces the memory requirements. Fourth is choosing the size of the Pooling Layers windows: the feature space is relatively limited and hence, the use of 4×4 pooling layers will not affect the accuracy meanwhile it will improve significantly the resources' requirements of the succeeding layers. Fifth is to avoid padding throughout the CNN because: a) does not affect the accuracy and b) given that the majority of the target applications have objects located at or close to the image center, is needless to preserve the size of the feature maps. Sixth is the divisibility of each convolution layer's output by the kernel size of the succeeding pooling layer: a) it allows the omission of padding with no accuracy loss and b) leads to efficiently pipeline these layers.

The second phase develops a bit accurate model (BAM) of the CNN with the weights of the first phase. The BAM has the exact same functionalities with the TensorFlow model and performs the exact same fixed-point calculations with the hardware accelerator.

The third phase maps the calculations of the BAM onto the FPGA by exploiting VHDL designed blocks implementing the key CNN functions. Each layer's design uses multiple instances of the following configurable and reusable VHDL blocks.

2.2. VHDL Blocks

Input Block: stores one RGB channel of a full image. It uses n shift registers called *Window Generator* with each register containing one image row. If the following Convolution Layer uses $n \times n$ kernels, the n shift registers forward an input $n \times n$ window per cycle to fully pipeline the two layers. Configurable are the: a) image size, b) the n registers, c) the kernel $n \times n$ and d) pixel bit depth.

Convolution Block: receives a single channel of the input image (or feature map) in the format of kernel sized windows and it calculates the convolution of all the filters in a pipeline fashion.

Pooling Block: from the $k \times k$ feature map of the preceding Convolution Layer it selects the maximum value of each $l \times l$ window, for all the windows with stride l and outputs the $(k/l) \times (k/l)$ array.

Vector Multiplier. It realizes a Fully Connected Layer neuron: computes the dot product of the 1-D vector flattened result of the preceding layer, one point at a time, and one row of the weight matrix.

The *ReLU Block* is a 2-to-1 multiplexer.

Output Block. It is the CNN's final Fully Connected Layer: it executes the matrix multiplication of the flattened array result I with the weights W and then adds the Bias. In a pipeline fashion is executed once for each output neuron/class.

2.3. Methodology for Mapping the CNN on the FPGA

The methods for improving the key issues of the FPGA accelerator are:

Buffers between layers and Speed-up: The effort is given to parallelize the N filters in each Convolution Layer because it maximizes the speed-up, allows to pipeline the input to all the filters and avoids the buffer between this and its preceding layer.

The memory of each layer: each Convolution Layer produces N feature maps and apart the first accumulates these in N memories. The size of each of these N memories depends on the size of the preceding layers. If the input image has size $Q \times Q$ and there are p pooling layers of sizes $sp_i \times sp_i$, $1 \leq i \leq p$, each memory has size $[Q \times Q] / \prod_{i=1}^{i=p}$. Hence, higher dimension pooling layers reduce the memory size and allow to implement N parallel filters with their individual memories.

The First Convolution Layer. To reduce memory requirements this layer parallelizes only the RGB channel computations by using 3 convolution blocks, one block per channel. A tree of adders accumulates each channel's convolution result and forwards it to the following Pooling Layer without buffer. If the following pooling layer has size $sp_0 \times sp_0$, this layer will be slower by $(sp_0)^2$. If there are resources, we can use k ($k \leq (sp_0)^2$) such structures in parallel to improve the speed-up by k.

Scalability. The Fully Connected Layers can use a *Vector Multiplier* per neuron: parallelizing the neurons is advantageous leading to a layer design irrespective of the size and the number of the feature maps produced by the preceding layer; and more importantly it is scalable.

3. Case Design: Vessel Detection CNN Accelerator

The following paragraphs employ the approach to develop a Vessel Detection FPGA accelerator that can also be used in sliding window applications of large images.

3.1. Model Architecture and Training

The model was trained with the "Ships in Satellite Imagery" Kaggle dataset. It contains 4000 80x80 RGB images in total, labeled with either "ship" or "no-ship" binary classification: 3K images were selected for the training process and the remaining for model validation.

A variety of training processes was performed with the TensorFlow Estimator API in Python to create a CNN model close to optimal with respect to prediction accuracy, number of operations and resources requirements. The CNN model consists of 84K weights optimized using the Adam optimizer with the cross-entropy loss function; it achieves 97.6% accuracy after 50 epochs. It compares favorably to similar trained models due to the following results of the design study.

Number of Convolution layers: The proposed CNN with only 2 convolution layers achieves accuracy 97.6% that is close to CNNs with more, e.g. a CNN with 3 convolution layers, before any of the proposed optimizations achieved 98.5% accuracy.

Ship Orientation: is limited and along with the proportion of the 80x80 image that the ship occupies, it leads to use 32 filters per convolution layer for achieving the best accuracy-computational cost trade-off.

Max Pooling layer: size 4×4 reduces the feature map's size and achieves accuracy similar to that of 2×2.

The kernel's size for each Convolution Layer: the first achieved improved accuracy with a 5×5 kernel, while the Second with 4×4 because its output has to be divisible by the following Max Pooling layer in order to avoid padding, since this doesn't induce accuracy loss.

Fully Connected Layer's neurons: 128 is the minimum that avoids prediction accuracy loss.

3.2. Bit Accurate Model (BAM)

The BAM represents each parameter of the CNN model (weights, biases) and pixel values of each input image channel (RGB) as a fixed-point number with 1-bit for the sign, 1-bit integer part and 6-bit fractional part (Q2.6). Each channel uses 8 bits in Q2.6 representation and throughout the BAM we preserve the 6-bit fractional part by truncating the result of each multiplication. In order to avoid accuracy losses due to overflow after consecutive additions the integer part is increased and the final results are represented in Q11.6.

3.3. FPGA Accelerator

The architecture in fig. 1 uses the blocks described in Section 2.2: the Input Layer with the *Window Generators* and the First Convolution Layer including three *Convolution Blocks*. Their output is forwarded to the ReLU and the First Pooling Layer consisting of one Pooling Block configured for 4×4 max pooling. The Second Input Layer includes a single input block. The second Convolution Layer includes 32 Convolution Blocks and the corresponding accumulators followed by the second pooling layer. Finally, the data are forwarded to the Fully Connected Layer, which uses 128 parallel *Vector Multipliers*. 128 parallel multipliers implement the functions of the output layer.

The input block supports the FPGA's PCIe, Ethernet and USB to receive the image. Each input image channel is stored row by row in the *Channel Block RAM*, so that we can read a whole row in a single clock cycle. Three distinct RGB windows of size 5×5 forwarded in parallel at each clock cycle to the *Convolution Blocks* in a fully pipelined operation.

4. FPGA Results & Comparison to Edge Devices

The development and validation of the Accelerator targeted the Xilinx Virtex 7 board with Vivado. The resource utilization of the FPGA is 16.71% LUT, 3.23% LUTRAM, 11.66% FF, 9.37% BRAM and 30.11% DSP blocks. The Accelerator power requirements are 5.001 W and it has achieved a maximum operating frequency of 270 MHz. The processing time for a single input image (or 80×80 sliding window) is 0.687 ms, compared to the 76.9 ms of the single threaded code on an Intel i7-8750H achieving speed-up of 111.93×. Moreover, the execution time on an NVIDIA GTX 960 GPU is 14.6ms; the FPGA Accelerator has a 21.25× speed-up over the GPU. The number of operations is 52.8 GOPS.

In order to evaluate the proposed approach we compared the performance of the Vessel Detection CNN FPGA Accelerator to the other edge devices, the NVIDIA's Jetson Nano (128-core Maxwell GPU, 10W) and the Intel's Myriad2 processor (2 Leon & 12 SHAVE processors, 512MB low-power DDR3, 2MB multicore on-chip memory subsystem CMX, 1W). The comparison is based on a sequential C code for the vessel detection and a custom CUDA accelerated application.The mapping of calculations to grids of thread blocks optimize the scheduling of warps on the 128 Cuda cores. The development on the Myriad2 starts with the optimization of the sequential C code, using efficiently the CMX, DDR and cache memories. The parallel Myriad2 code uses the 12 SHAVES, by dividing the CMX memory between them, minimizing the required memory of each SHAVE by pipeling the operations of each processor, the parallel code takes 14.6 ms at 1 W.

The detailed results are presented in Table 1. The FPGA accelerator achieves the highest performance, regarding execution time, median power consumption but the highest performance per watt. The Myriad2 is the most power efficient by consuming 1 W, while its performance is one order of magnitude lower than the FPGA. The Jetson Nano falls short in either metric but it has the most developer friendly platform.

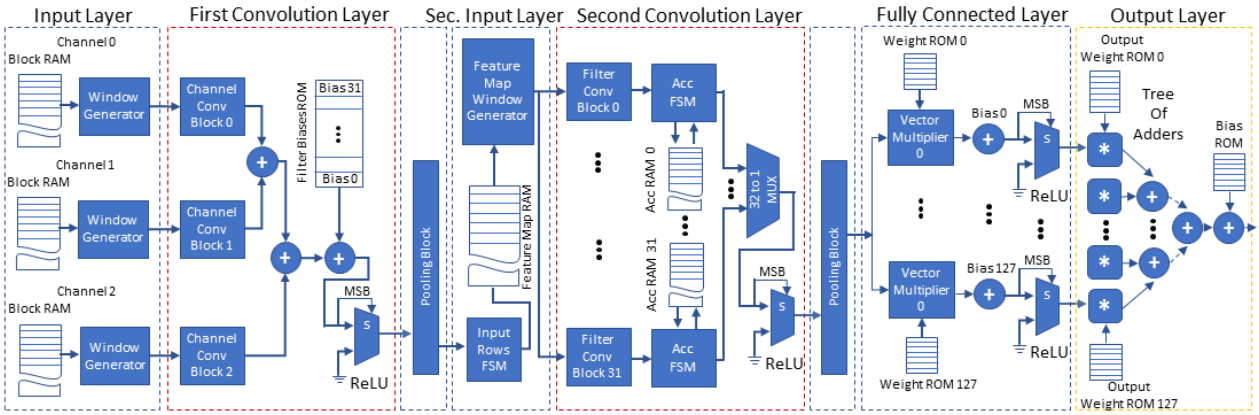

Figure 1. The Architecture of the FPGA Accelerator.

Finally, the single device approach improves performance compared to optimized FPGA CNN accelerators [8] and also to low cost ones [7].

Table 1
Performance & Power Comparison to Edge Devices

	Execution Time (ms)	Speed-Up	Power (W)
Jetson Nano CPU	440	-	10
Jetson Nano GPU	20.3	21.7	10
Myriad2 1 SHAVE	56.27	7.8	0.5
Myriad2 12 SHAVE	14.59	30.1	1
FPGA Accelerator	0.687	640.5	5

5. Conclusion

The current paper presented a design approach for FPGA Accelerators for CNNs with limited feature space, for edge and mobile applications. It targets real-time performance by placing all the CNN computations and memory within a single FPGA device. The benefits of the resulting architecture are the low power consumption, the operating frequency and the resources utilization. Finally, the FPGA example design for the Vessel Detection compares favorably to the performance of notable edge processors.

Acknowledgement

HFRI This study was funded in part by the HFRI PhD Fellowship grant (No.: 29).

References

[1] P. Sermanet, Y. LeCun, Traffic sign recognition with multi-scale convolutional networks, in: The 2011 International Joint Conference on Neural Networks, 2011, pp. 2809–2813. doi:10.1109/IJCNN.2011.6033589.

[2] Airbus ship detection challenge (2019). URL https://www.kaggle.com/c/airbus-ship-detection

[3] O. Gorokhovatskyi, O. Peredrii, Shallow convolutional neural networks for pattern recognition problems, 2018. doi:10.1109/DSMP.2018.8478540.

[4] Planet: Ships-in-satellite-imagery (2019). URL https://www.kaggle.com/rhammell/ships-in-satellite-imagery

[5] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, V. Toma, Always-on vision processing unit for mobile applications, IEEE Micro 35 (2) (2015) 56–66. doi:10.1109/MM.2015.10.

[6] Nvidia jetson nano (2020). URL https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[7] R. A. Solovyev, A. A. Kalinin, A. G. Kustov, D. V. Telpukhov, V. S. Ruhlov, Fpga implementation of convolutional neural networks with fixed-point calculations, CoRR abs/1808.09945 (2018).

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing fpga-based accelerator design for deep convolutional neural networks, in: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '15, ACM, New York, NY, USA, 2015, pp. 161–170. doi:10.1145/2684746.2689060.

[9] M. Peemen, A. A. A. Setio, B. Mesman, H. Corporaal, Memory-centric accelerator design for convolutional neural networks, in: 2013 IEEE 31st International Conference on Computer Design (ICCD), 2013, pp. 13–19. doi:10.1109/ICCD.2013.6657019.

[10] S. Ren, K. He, R. B. Girshick, J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks, CoRR abs/1506.01497 (2015). arXiv:1506.01497.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, A. C. Berg, SSD: single shot multibox detector, CoRR abs/1512.02325 (2015). arXiv:1512.02325.