Optimizing Temporal Convolutional Network inference on FPGA-based accelerators

Marco Carreras, Gianfranco Deriu, Luigi Raffo, Luca Benini and Paolo Meloni

***Abstract*—Convolutional Neural Networks are extensively used in a wide range of applications, commonly including computer vision tasks like image and video classification, recognition and segmentation. Recent research results demonstrate that multi-layer (deep) network involving mono-dimensional convolutions and dilation can be effectively used in time series and sequences classification and segmentation, as well as in tasks involving sequence modelling. These structures, commonly referred to as Temporal Convolutional Networks (TCNs), have been demonstrated to consistently outperform Recurrent Neural Networks in terms of accuracy and training time [1]. While FPGA-based inference accelerators for classic CNNs are widespread, literature is lacking in a quantitative evaluation of their usability on inference for TCN models. In this paper we present such an evaluation, considering a CNN accelerator with specific features supporting TCN kernels as a reference and a set of state-of-the-art TCNs as benchmark. Experimental results show that, during TCN execution, operational intensity can be critical for the overall performance. We propose a convolution scheduling based on batch processing that can boost efficiency up to 96% of theoretical peak performance. Overall we can achieve up to 111,8 GOPS/s and a power efficiency of 33,9 GOPS/s/W on an Ultrascale+ ZU3EG (up to 10x speedup and 3x power efficiency improvement with respect to pure software implementation).**

***Index Terms*—Temporal Convolutional Network, TCN, hardware accelerator, FPGA, embedded systems**

I. Introduction

The widespread use of Convolutional Neural Networks (CNNs), as a go-to solution for a wide range of AI-related problems, combined with the high computational load typically associated with their execution, has led researchers to extensively work on the development of hardware accelerators for CNN inference [2][3][4][5]. Availability of this kind of hardware is key in embedded use-cases involving near-sensor processing of data, according to the edge computing paradigm [6]. Among the different solutions available in literature, an important role is played by FPGA-based architectures, and, in more detail, by solutions exploiting the cooperation between general-purpose processors and FPGAs available in modern All-programmable SoCs, e.g. Zynq and Zynq Ultrascale+ devices by Xilinx, that take profit from the efficient implementation of Multiply-And-Accumulate (MAC) operations on the large amount of DSP Slices available [7]. CNNs have been employed for computer vision applications, such as image recognition [8], [9], [10], object detection [11] or video frame classification [12]. However, convolutional networks have also been extended to deal with time sequences. These CNN variations are usually indicated as Temporal Convolutional Networks (TCNs) [13]. Multiple TCN architectures have been proposed, reaching impressive performance on tasks such as sentence classification [14], speech recognition [15], text understanding [16], Natural Language Processing tasks [17] and, more recently, machine translation [18], audio synthesis [19], language modeling [20] or signal sequence analysis in the healthcare domain, such as action detection [21] or ECG classification [22]. Research work of Bai et al. [1] demonstrates that the exploitation of a Temporal Convolutional Network for typical sequence modelling tasks often outperforms older and better-known Recurrent Neural Networks [23] [24].

Thus, the rapidly increasing interest in TCNs pushes for investigating on acceleration for these networks. Especially in the embedded domain, where (near) real-time analysis of sequences of data samples acquired by sensors is a common case, accelerating this kind of workload on reconfigurable devices is a very appealing approach. In this work we explore the capabilities of a state-of-the-art CNN inference accelerator [25] specifically enriched to provide the flexibility needed in TCNs, to support freely selectable *kernel sizes* and *dilated* convolutions, with freely selectable *dilation rates*. We focus on an implementation on low-cost and low-power all-programmable SoCs, more suitable for the integration of edge-computing and IoT processing nodes, considering two widely accessible devices in the Zynq and the Zynq Ultrascale+ families. Performance is evaluated over various benchmark TCNs, reporting on absolute execution time as well as on the efficiency of the execution with respect to the peak performance imposed by the device resources. We propose an optimization method relying on *batch processing* to improve efficiency in cases where operational intensity is critical. We also assess the efficiency of the FPGA-based acceleration, comparing with software execution and with state of the art accelerators on bi-dimensional CNNs.

The reminder of this paper is organized as follows: Section II discusses the related work. Section III analyses the TCN model. Section IV gives an overview of the accelerator architecture taken as reference for this work. Section V describes implementation details and strategies. The experimental results are presented in Section VI and Section VII. Conclusions are exposed in Section VIII.

M. Carreras, G. Deriu, L. Raffo and P. Meloni are with the Department of Electrical and Computer Engineering, Università degli Studi di Cagliari, Cagliari, Sardinia, 09123 Italy (e-mail: marco.carreras@unica.it, gianfranco.deriu@unica.it, paolo.meloni@unica.it, raffo@unica.it).

L. Benini is with Università di Bologna, , Bologna, Italy and ETHZ, Zurich, Switzerland (e-mail: luca.benini@unibo.it and lbenini@ethz.ch).

This work has received funding from the European Unions Horizon 2020 Research and Innovation Programme under grant agreement No. 780788.

II. Related Work

The landscape of FPGA-based accelerators for CNN is crowded and multifaceted [26]. Several approaches have been proposed in recent years, focusing both on the embedded domain and on architectures aimed to speed-up execution on cloud servers. However, work on FPGA acceleration has mainly focused on classic CNN networks.

Yu et al. [27] developed an FPGA acceleration platform that leverages a unified framework architecture for general-purpose CNN inference acceleration at a data center achieving a throughput comparable with the state-of-the-art GPU in this field, with less latency. This work exploits on-chip DSPs, on a Kintex KU115, arranged as supertile units (SUs), to overcome the computational bound and, together with dispaching-assembling model and broadcast caches, to deal with the memory bound.

Zhang et. al. [28] proposed Caffeine, a hardware/software library to efficiently accelerate CNNs on FPGAs, leveraging a uniformed convolutional matrix multiplication representation targeting both computation-intensive convolutional layers and communication-intensive fully connected layers of CNN which maximizes the underlying FPGA computing and bandwidth resources utilization. Ma et. al. [29] presented an RTL-level CNN compiler that generates automatically customized FPGA hardware for the inference tasks of CNNs from software to FPGA. The approach proposed by [29] relies on a template accelerator architecture described in Verilog including all the main functions employed by CNNs such as convolutions, pooling, etc, which are automatically customized at design time to match the requirements of the target CNN model. The proposed methodology is demonstrated with end-to-end FPGA implementations of complex CNN models such as NiN, VGG-16, ResNet-50, and ResNet-152 on two standalone Intel FPGAs, Stratix V and Arria 10, providing average performance up to 720 GOps.

These two frameworks provide huge performance gains when compared to state-of-the-art accelerators and general-purpose CPUs and GPUs. However, both leverage large FPGA devices such as Virtex7 and Arria 10, they mainly target server applications exploiting batching to improve memory access performance and bandwidth utilization.

This approach is less suitable for embedded applications where cheap and compact SoCs integrating embedded processors and FPGAs are desirable, and images have to be processed in real-time. In this domain, Venieris et. al. [30] presented a latency-driven design methodology for mapping CNNs on FPGAs. As opposed to previously presented approaches mainly intended for bandwidth-driven applications, this work targets real-time applications where the batch size is constrained to one, relying on Xilinx high-level synthesis tools for mapping (i.e. Vivado HLS), demonstrated on relatively simple CNN such as AlexNet, and on a very regular one such as VGG16 featuring only 3×3 kernels, providing a peak performance of 123 GOps. Other work focuses on a template-based approach based on programmable or customizable RTL accelerators proposed in architectures [29][31][32], more similar to the one that is used in this paper.

SnowFlake [31] exploits a hierarchical design composed of multiple compute clusters. Each cluster is composed of four vectorial compute units including a vectorial MAC, vectorial max, a maps buffer, weights buffers and trace decoders. SnowFlake provides a computational efficiency of 91%, and an operating frequency of 250 MHz (best-in-class for CNN accelerators on Xilinx Zynq Z-7045 SoC). However, although the vector processor-like nature of the accelerator is very flexible, delivering significant performance also for 1×1 kernels, it prevents to fully exploit of spatial computation typical of application-specific accelerators, which leads to overheads due to load/store operations necessary to fetch weights and maps from the buffers. This is highlighted by the low utilization of the DSP slices available on the FPGA (i.e. only 256 over 900), and by the performance when executing end-to-end convolutional neural networks, which is lower than that of other architectures including the proposed one even though the operating frequency of the CNN engine is significantly higher.

Several approaches tackling FPGA architectures for image-processing CNN, have explored the reduction of the precision of arithmetic operands to improve energy efficiency. Although most of the architectures available in literature feature a precision of 16-bit (fixed-point)[30], [31], [29] numerous reduced-precision implementations have been proposed recently, relying on 8-bit, 4-bit accuracy for both maps and weights, exploiting the resiliency of CNNs to quantization and approximation [32].

Qiu et. al. [32] proposed a CNN accelerator implemented on a Xilinx Zynq platform exploiting specific hardware to support 8/4 bit dynamic precision quantization, at the cost of 0.4% loss of classification accuracy. Other extreme approaches to quantization exploit ternary [33] or binary [34] neural-networks accelerators for FPGA. This approach significantly improves the computational efficiency of FPGA Accelerators, allowing to achieve performance level as big as 8 TOPS [33].

Recent work by Rasoulinezhad et al. [35], starting from the Xilinx DSP slices, proposed an optimized DSP block called PIR-DSP to efficient map 9, 4 and 2 bits data precision MAC operations. It is implemented as a parameterized module generator targeting both FPGAs and ASICs reaching an estimate run time energy decrease up to 31% for a MobileNet-v2 implementation compared with a standard DSP mode. Other works, like Wang et al. [36], leverage FPGA LUT blocks as inference operators for Binary Neural Network (BNN) achieving up to twice area efficiency compared to state-of-the-art binarized NN implementation and against several standard networks models.

While small networks like MNIST, CIFAR10, SVHN, GTSRB can reach good classification accuracy, the training is still a big challenge for larger networks such as VGG or ResNet [37]. The usability of extreme quantization is also not demonstrated for TCN-related tasks, sequence classification and modelling.

Probably the most powerful currently available FPGA-based acceleration engine is the proprietary one offered by Xilinx, which provides an integrated framework, called VitisAI [38], that helps designers in mapping CNNs on a templated soft IP

called Deep Learning Processing Unit (DPU) [39]. The DPU provides impressive performance on CNNs, using quantization and high clock frequency in DSP slices. Quantization is required and can be applied automatically using a dedicated tool included in VitisAI. DSP slices are clocked at very high frequency, using a *DSP Double Data Rate (DDR) technique* [40], which uses a 2x frequency domain to increase peak performance. However, the support for TCN is missing, since the DPU does not support arbitrary dilation, stride and kernel sizes and the the VitisAI quantization process does not support 1D convolutions. To the best of our knowledge, there are no published FPGA-based accelerators tuned to speed-up inference for generic Temporal Convolutional Networks.

As main novel contributions of this work, we propose:

- an enriched architectural template supporting efficient TCN inference on FPGA;
- a methodology for the optimal execution/scheduling of data-transfers exploiting the specific sequence-based structure of data in TCNs
- a methodology for improving efficiency based on *sample batching* (sequence buffering)
- the first (to the best of our knowledge) experimental evaluation of the usability of FPGA-based acceleration for TCNs, based on different end-to-end benchmarks and two APSoC devices

III. TCN MODEL GENERALITIES

A dilated convolution operation F on element s of a sequence [1] can be defined as:

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i} \qquad (1)$$

where $x \in \mathbf{R}^n$ is a 1-D input sequence, $f : \{0, ..., k-1\} \in \mathbf{R}$ is a kernel of size k and d is the $dilation\ rate$.

The sequence of samples that constitute the input of a TCN can be processed both off-line or in a real-time streaming. The second case is very useful in application cases requiring continuous analysis of the input sequence, e.g. aimed at the identification of specific events and/or at promptly closing the loop on data-triggered actuations. This implies that the sequence must be analyzed at every time step, after being updated with a new sample. It is possible to identify the minimal sequence size to produce a valuable output sample as the *receptive field* that depends on convolutional layer parameters such as the *kernel_size* and the *dilation rate*:

$$receptive field = 1 + \sum_{l=1}^{L} [k(l) - 1] \times d(l) \qquad (2)$$

where $l \in 1, 2...L$ is a layer of the network. This can be thought of as a sliding processing window for the input sequence.

Figure 1 highlights these concepts. It is worth noticing that increasing the dilation parameter leads to an increase of the network's memory without affecting its depth.

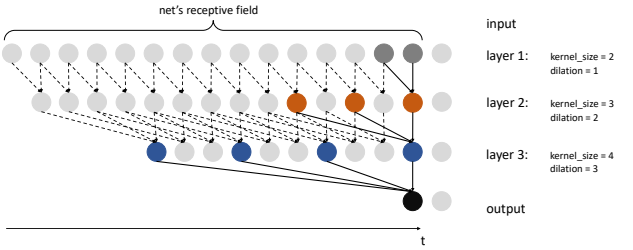


Fig. 1: TCN execution graph. For this example, see (2), $receptive field = 1 + (2-1) \times 1 + (3-1) \times 2 + (4-1) \times 3 = 15$

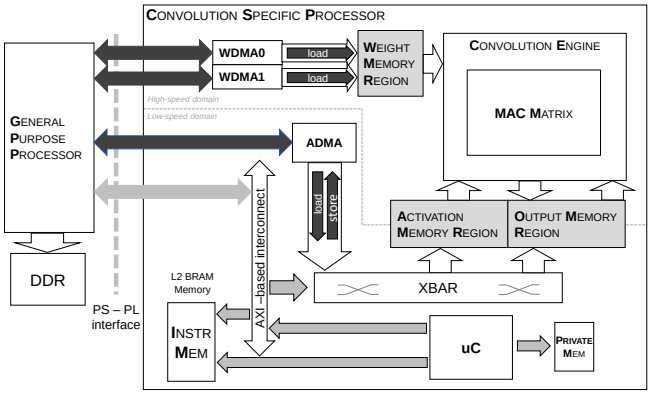


Fig. 2: NEURAghe architecture

IV. REFERENCE ACCELERATOR ARCHITECTURE

Our TCN accelerator is an extension of NEURAghe [25], a CNN inference accelerator architecture that exploits the cooperation between the ARM Cortex-A9 Processing System (PS) and the Programmable Logic (PL) in Xilinx Zynq devices. Communication at the PS-PL interface is allowed by the PS high-performance general purpose ports. In NEURAghe, as can be seen in Figure 2, the Programmable Logic hosts a Convolution Specific Processor (CSP), while the processing system acts as a General Purpose Processor (GPP) dealing with tasks hard to accelerate by parallelization on programmable logic, such as data marshalling or non-Conv layers. NEURAghe integrates a RISC-V lightweight processor inside the CSP, dedicated to the execution of a firmware that schedules data transfers and convolutions without PS intervention, leaving the latter available for actual computation workload. In order to support TCN execution, we have enhanced the CPS IP, described in System Verilog and available as open source [1]. We have added new features to improve flexibility, requiring substantial modification of the main computational core dedicated to Multiply and Accumulate (MAC) operations execution, called Convolution Engine, and an improvement of the circuitry and procedures managing transfers to/from the DDR memory. In particular, the Convolution Engine (Figure 3) is composed by a MAC Matrix of N_{cols} columns by N_{rows} rows of Sum of Product (SoP) units in charge to calculate the contribution of N_{cols} input features to N_{rows} output features. N_{rows} Shift Adder modules sum together partial result from

[1] https://github.com/neuraghe/NEURAghe

SoPs in each row with data values resulted from the N_{cols} previously computed input feature partial results read from on-chip memory, enabling successive accumulation over multiple CE runs.

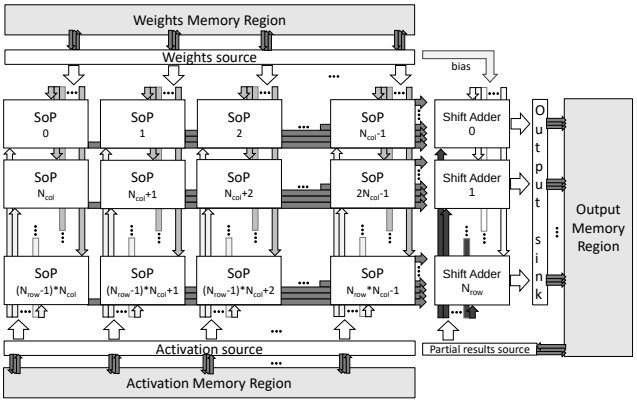


Fig. 3: Convolution Engine. $N_{rows} \times N_{cols}$ MAC Matrix

V. TCN SUPPORTING HARDWARE FEATURES

Considering the typical features of TCNs, we designed the CE according to several design principles. The accelerator:

- has to be $kernel\_size$ agnostic;
- must execute convolutions with multiple stride values without performance overhead;
- must support freely selectable dilation values.

A. Freely selectable kernel sizes

To support arbitrary kernel sizes, we have chosen to dedicate one single DSP cell to compute an entire convolution kernel, reusing it over a number of cycles depending on the kernel size. A new sample of the output feature under production is thus produced after $kernel\_size$ cycles and is ready to be sent to the Shift Adder module. Using one single DSP cell per kernel can easily require the instantiation of a very high number of SoPs, to the aim of exploiting as many resources as possible among those available on the target device. To keep the MAC Matrix growth feasible, we designed SoPs to be composed by 4 Xilinx DSP48E primitives, performing 4 $MACs/cycle$, operating in parallel on 4 different 16-bit samples, as visualized in Figure 4, from 4 neighbour convolution windows in an input feature.

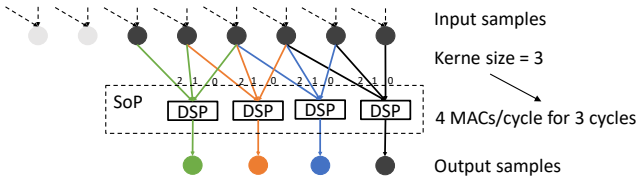


Fig. 4: SoP elaboration scheme

Figure 5 represents the organization of a SoP. Considering the template proposed in Figure 3, the MAC Matrix implementation requires a deterministic number of DSP48E primitives, as indicated by Equation 3.

$$N_{DSPs} = N_{rows} \times N_{cols} \times 4 \quad (3)$$

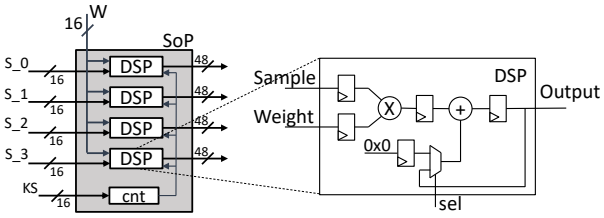


Fig. 5: Sum of Product Unit

B. Flexible activations and weights fetching

To enable arbitrary stride and dilation values, the fetching of input samples from the internal memory has been designed to be very flexible. Each CE port dedicated to input samples is endowed by a programmable Activation Source module, while weight kernels are fetched from Weight Memory banks by means of Weights Source modules. Such source modules can be programmed at start-up according to stride, dilation, aspect ratio and size of activations and weight kernels. Fetching of bi-dimensional memory sections is enabled, to support generic CNN execution.

The memory subsystem has been designed to enable conflict-less loading of neighbour convolution windows. The Activation Source module controls N_{cols} ports of the convolution engine. Each one loads samples from a dedicated BRAM module. Considering that four samples, belonging to different windows, can be loaded in the same cycle, to support stride values up to 3 samples (which is sufficient to support most of the TCN use-cases available in literature), each module of the activation memory has to be composed of at least 8 different independently accessible RAMB18 modules. Figure 7 represents an example where a different configuration of RAMB18 modules can determine a conflict.

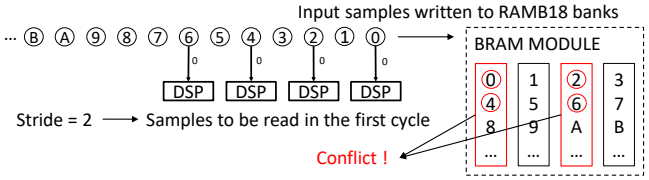


Fig. 7: BRAM read conflict example. Samples are stored in BRAM modules using interleaving. Consecutive samples are stored in adjacent RAMB18 banks. 4 DSP slices in a SoP units, with stride 2, in the first cycle of the convolution, load respectively samples 0,2,4,6. With four RAMB18 banks, samples 0-4 and 2-6 are on the same bank, creating a conflict.

Moreover, the Weight source module controls one port per each SoP in the matrix, which has to be implemented by at least one RAMB18 module.

Finally, the CE has a set of ports that are used to write results and to load previously computed partial results, when a convolution requires to accumulate over several accelerator operations. These ports are controlled by a Partial Result Source module and by an Output Sink module. Each of these modules controls N_{rows} ports, each one writing/reading

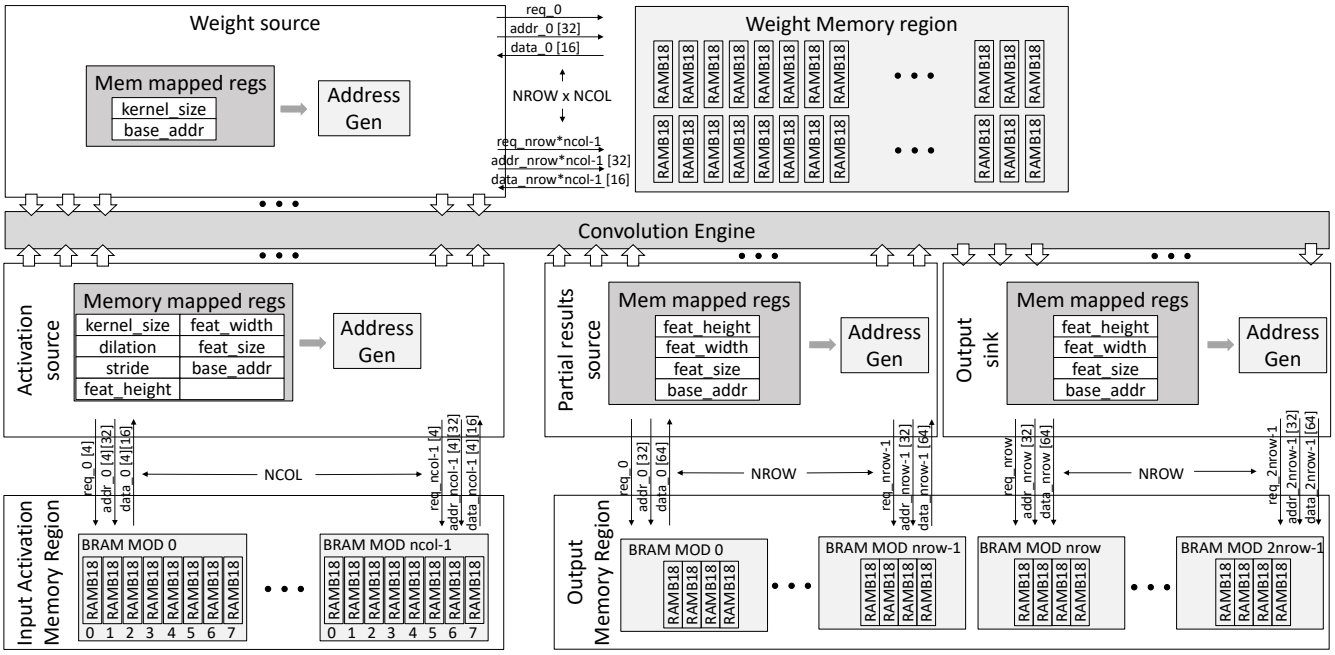


Fig. 6: Memory transfers

four samples simultaneously. Thus BRAM modules in the corresponding memory region are composed by at least 8 RAMB18 modules each.

Samples and weights, in the experiments presented in this paper, are all using a 16-bit data format, thus RAMB18 modules are configured to expose two 16-bit addressable ports and can be 1024 words deep.

Considering the described organization, a given architectural configuration requires a number of RAMB18 primitives that can be deterministically estimated as indicated in Equation 4.

$$N_{BRAMs} = N_{rows} \times N_{cols} + N_{cols} \times 8 + (N_{rows} \times 2) \times 8 + 32 \quad (4)$$

The first component corresponds to weight memory, the second to the modules storing input activations, the third to output and partial results memories. 32 blocks are used to implement the RISC-V scheduler instruction memory and private memory.

C. TCN support in firmware

The programming model used in NEURAghe envisions the ARM-based processing system in the Zynq platforms to execute a C program implementing the neural network inference. When the accelerator implemented in the programmable logic has to be used. The program in the PS sends commands describing the layer to be executed. The RISC-V soft-core in the accelerator decodes the command and decomposes the layer in sub-operations, namely partial convolutions in the accelerator and data transfers from/to the off-chip memory, executing an optimized firmware which is also coded in C. The firmware uses a double-buffering technique, to allow the accelerator to overlap transfers phases with convolutions, as represented in Figure 8, reducing as much as possible idle times in the CE to exploit the processing capabilities of DSP slices with maximum efficiency.

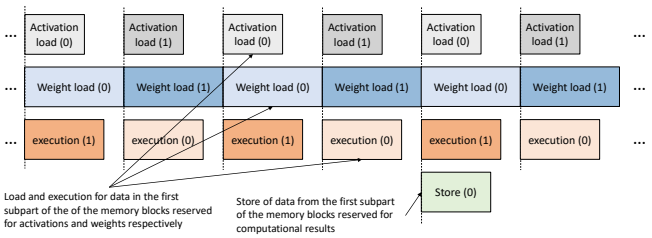


Fig. 8: Scheduling scheme

When considering TCN executions, the previously described paradigm has to be applied to the characteristics of the algorithm. For every Convolutional Layer in a TCN, given its *kernel_size* and a *dilation*, it is possible to consider a local *receptive field*, indicated as RF_{local} in Equation 5, that is the minimum amount of layer's input samples per channel needed to produce a valuable output sample.

$$RF_{local} = 1 + (k-1) \times d \quad (5)$$

Figure 9a shows how input and output transfers are implemented for TCNs. At every new time step, input to a layer is updated by adding one new sample to input features. In order to execute the layer, the firmware triggers DMA transfers to load to the activation input memory RF_{local} samples per each input feature. After the execution, one output sample is produced per output feature. Output samples are sent to DDR using an output DMA transfer, to be stored until the next time step.

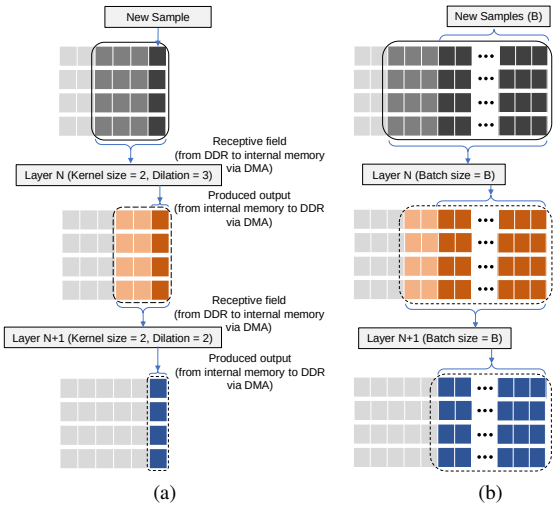


Fig. 9: TCN execution on NEURAghe

D. Improving through batch processing

Although the previous approach provides the minimum classification/recognition latency, executing the network every time a new sample is available to update the input sliding window, it can determine performance to be bandwidth limited. This is because all of network parameters/weights must be loaded for every layer. So, despite the double-buffered scheduling strategy, transfer and computation phases hardly overlap, affecting the *operational intensity* of the application. The roofline model in Figure 10 shows performance trend

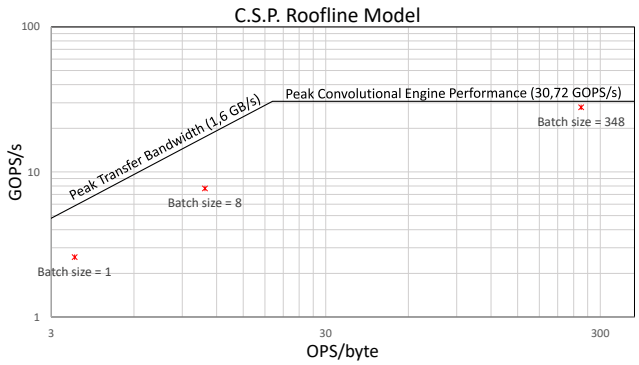


Fig. 10: Use case benchmark's Roofline Model for the Convolution Specific Processor (12x4 MAC Matrix in Z-7020 SoC) with respect to different batch sizes

starting from the sample-by-sample processing (leftmost red cross), on a use-case that will be presented in the following.

If the latency constraint is not extremely tight, it is possible to pre-buffer input samples in order to process longer sample sequences (batch) and produce more output with every execution. Figure 9b shows the transfer scheduling when *batch size* is increased. As the *batch size* increases the architecture gets closer and closer to the computational limit (rightmost red cross on the roofline plot of Figure 10) because of the growing number of operations performed. In this way, it is possible to increase the utilization of computing resources and to gain efficiency. As may be noticed in Figure 10, when sample batch size is small, besides being on the badwidth-limited region of the plot, points in the roofline model are also distant from the theoretical achievable performance. This is due to overheads related with CE programming/warm-up, and to initial and final input/output transfers, which cannot overlap with convolutions. The impact of this overhead is limited when the operational intensity increases, reducing the distance between points and the theoretical roofline model.

VI. Hardware Implementation Evaluation

A. Design Space Exploration

A designer willing to use the NEURAghe template, on a given target SoC, has multiple possible architectural configurations available. In order to perform a careful selection, it is possible to perform a simple design space exploration and to choose a near-optimal setup from the performance point of view. To select the architectures presented in this paper, we have used a simple grid search that evaluates multiple configurations, featuring different values of N_{rows} and N_{cols}, to maximize the number of SoPs, while keeping the number of used DSPs and BRAMs in the range of availability imposed by the target SoC. Table I shows utilization numbers estimated using the Equations 3 and 4. Light-gray coloured cells in the table indicate which configurations are not implementable in a Z7020 SoC, due to excessive DSP or BRAM utilization. Darker-coloured cells indicate configurations that are not feasible in an Ultrascale+ ZU3EG device.

TABLE I: RAMB18 and DSP utilization in NEURAghe architecture with respect to the MAC Matrix shape

RAMB / DSP

N_{rows} \ N_{cols}	4	5	6	7	8	9	10	11	12
4	144 / 64	156 / 80	168 / 96	180 / 112	192 / 128	204 / 144	216 / 160	228 / 176	240 / 192
5	164 / 80	177 / 100	190 / 120	203 / 140	216 / 160	229 / 180	242 / 200	255 / 220	268 / 240
6	184 / 96	198 / 120	212 / 144	226 / 168	240 / 192	254 / 216	268 / 240	282 / 264	296 / 288
7	204 / 112	219 / 140	234 / 168	249 / 196	264 / 224	279 / 252	294 / 280	309 / 308	324 / 336
8	224 / 128	240 / 160	256 / 192	272 / 224	288 / 256	304 / 288	320 / 320	336 / 352	352 / 384
9	244 / 144	261 / 180	278 / 216	295 / 252	312 / 288	329 / 324	346 / 360	363 / 396	380 / 432
10	264 / 160	282 / 200	300 / 240	318 / 280	336 / 320	354 / 360	372 / 400	390 / 440	408 / 480
11	284 / 176	303 / 220	322 / 264	341 / 308	360 / 352	379 / 396	398 / 440	417 / 484	436 / 528
12	304 / 192	324 / 240	344 / 288	364 / 336	384 / 384	404 / 432	424 / 480	444 / 528	464 / 576

	avail. RAMB18	avail. DSP	selected	out of resources
Z7020	280	220		
ZU3EG	432	360		

B. Implementation on different SoCs

We have implemented the NEURAghe configurations with different Convolution Engine's MAC Matrix shapes, selected after the DSE process, on two target platforms: a Xilinx Zynq Z-7020 and a Xilinx Zynq Ultrascale+ ZU3EG. For the Z-7020 we have selected two configurations, featuring a similar number of DSP slices and similar clock frequencies. Table II shows resource occupation on the reconfigurable logic of a first configuration implemented on the Z-7020, featuring a 12×4 MAC matrix of SoP units, that can be clocked up to 120 MHz, providing peak performance of 46 GOPS/s. Table III shows results related with a similar configuration that integrates an 11×5 MAC matrix, using slightly more DSPs but clockable up to 110 MHz, with a peak performance of 48.4 GOPS/s. Finally, Table IV describes the results of the implementation of a configuration, featuring a 9×10 MAC matrix of SoP modules, on the ZU3EG. This design uses all the DSP slices on the chip and can be clocked at 180 MHz, providing a peak of 129.6 GOPS/s.

TABLE II: Resource occupation on a Xilinx Zynq Z-7020 (12x4 MAC matrix)

	DSP	BRAM	LUTs (logic)	LUTs SR	Regs
Used	192	120	47230	259	26942
Available	220	140	53200	53200	106400
%	87.27	85.71	88.78	1.49	25.32

TABLE III: Resource occupation on a Xilinx Zynq Z-7020 (11x5 MAC matrix)

	DSP	BRAM	LUTs (logic)	LUTs SR	Regs
Used	220	128	42964	283	26962
Available	220	140	53200	53200	106400
%	100	91.4	80.76	1.63	25.34

TABLE IV: Resource occupation on a Xilinx Zynq UltraScale+ ZU3EG (9x10 MAC matrix)

	DSP	BRAM	LUTs (logic)	LUTs SR	Regs
Used	360	354	47857	159	26463
Available	360	432	70560	70560	141120
%	100	81.94	67.82	0.4	18.75

VII. Experimental Results

We tested the actual level of performance that can be achieved using the accelerator on three different benchmarks, trying to cover as much as possible the landscape of TCNs presented in literature:

- a *plain* TCN network for ECG monitoring and classification (Goodfellow et al. [22]), that performs classification over single lead ECG waveforms and reaches around 90% average accuracy. We call this benchmark *ECG* in the following. This benchmark exposes real-time constraints and can be used to assess the usability of the accelerator in this kind of context.
- A network, called hereafter Res-TCN, based on residual units, with a structure taken from the Resnet family presented in [41]. The TCN is presented in ([21]) authors started from a 3D human action recognition dataset with 3D full skeleton annotations to extract a 1D feature representation per frame resulting in a 150-dimensional vector.
- A more complex network, based on WaveNet (Van DenOord et al. [14]), for Polyphonic Note Transcription of Time-Domain Audio Signal ([19]). We indicate this use-case as WN-PNT.

We have considered three system implementations using NEURAghe, two of them, 11×5 and 12×4, implemented on a Zedboard development board, with the MAC matrix respectively clocked at 70 and 80 MHz, and one implemented on the Ultra96 development board, with MAC Matrix clocked at 180 MHz.

Table V shows the comparison between on-chip memory capabilities for each system implementation and the memory footprint of each use-case network that regards the overall occupation of weight kernels and input features through all layers, considering the sample batch size values that will be used in the following. As may be noticed, all the networks must be adequately managed with the scheduling strategy presented in section V-C, since their activation and weight memory footprint exceeds the memory available on the considered low-cost hardware platforms.

TABLE V: Implementation related on-chip memory capabilities and use-case networks memory footprint

	input features [kB]			weight kernels [kB]
12x4 on Z-7020 on-chip memory	192			96
11x5 on Z-7020 on-chip memory	176			110
9x10 on ZU3EG on-chip memory	144			180
ECG:	**B=1**	**B=8**	**B=348**	
memory footprint	216,8	250,8	1696,5	11495,6
Res-TCN:	**B=1**		**B=144**	
memory footprint	37,3		511,5	5424,8
WN-PTN:	**B=1**		**B=504**	
memory footprint	553,5		5846,49	3328,8

A. ECG classification use-case

The network consists of several computational blocks mostly made of a 1D Convolutional layer, a batch normalization layer, a ReLU and a dropout stage, operating on streams of 16-bit samples, acquired at 300 Hz frequency. In this case, we have assessed three different operating modes:

- *minimum latency mode*,
- *maximum throughput mode*,
- *real-time execution*.

The first one provides classification in output after minimum latency. The network is executed as soon as possible per every input sample. In the second operating mode, sample batching is used extensively, to maximize throughput without considering latency as an optimization objective. In the third mode, sample batching is exploited just enough to reach

a throughput that allows respecting the real-time constraint posed by the input sampling frequency (300 Hz).

Figure 11 shows the performance achieved on this benchmark by configurations implemented on the Zedboard, while Figure 12 shows execution times.

As expected, without sample batching, performance is significantly bandwidth-limited. In minimum latency mode ($B = 1$ in the Figure), efficiency is very low for every layer, and consequently on the whole network. DSPs have very long idle times and actual performance is very far from the peak. Execution time is around 17 ms with the 12×4 matrix, and a bit higher with the 11×5.

To design the maximum throughput mode, we have iteratively tested different batch size values, to identify which value saturates the benefits achievable with sample batching. For this benchmark, such value is B=348, corresponding to a latency of around $1.2s$. In this case, both configurations can operate very near to the peak performance, with an average efficiency of around 0.9. Some layers, especially Type 1, are still less efficient, however, their contribution to the overall execution time is limited. In maximum throughput mode, each network execution takes around 140 ms, producing in output classification of 348 sample consecutive sample sequences, corresponding to one sample every $0.4ms$.

Finally, the real-time constraint requires to process one sequence in less than 3.3 ms. To design the real-time operating mode, by exploring the batch size, we identified $B = 8$, to be the lowest value enabling that respects such requirement. Execution time is around 17 ms, corresponding to around 2 ms per sample.

Since 12×4 is more efficient than 11×5 in all modes, in this case higher frequency is more important than the number of MACs, due to better utilization with the specific layer characteristics (the matrix is partially unused in the final operations of a convolution when the number of output features is not a multiple of 5).

TABLE VI: Convolutional Layer characteristics for Network ECG [22]

	type 1	type 2	type 3	type 4	type 5	type 6	type 7	type 8
$inputfeatures$	1	320	256	256	128	128	128	64
$outputfeatures$	320	256	256	128	128	128	64	64
$Kernel\_size$	24	16	16	8	8	8	8	8
$dilation\ rate$	1	2	4	4	6	8	8	8

The results are similar on the Ultra96 board, with increased performance, obviously, due to the higher clock frequency and the higher number of SoP modules. Efficiency and execution time are shown respectively in Figure 13 and in 14. Using minimum latency, efficiency is very low due to the bandwidth limits. Using $B = 348$ we have around 0.9 efficiency corresponding to around 0.1 ms per sample. To respect real-time constraint with minimum latency we can use $B = 4$.

B. Res-TCN use-case

This use case considers execution of the network presented in [21], made of stacked residual units, which perform 1D convolutions. Table VII shows the characteristics for the different types of Convolutional Layer in the network.

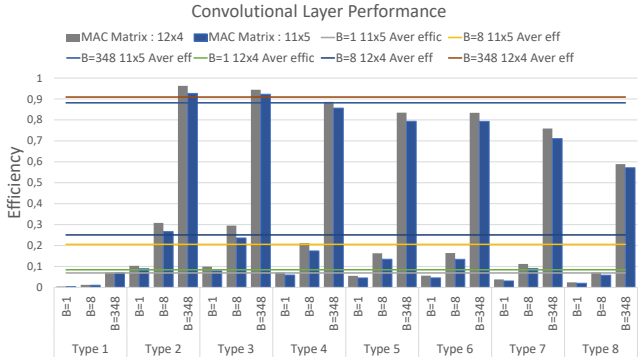


Fig. 11: Efficiency comparison on ECG [22] for NEURAghe 12x4 and 11x5 MAC matrix configurations

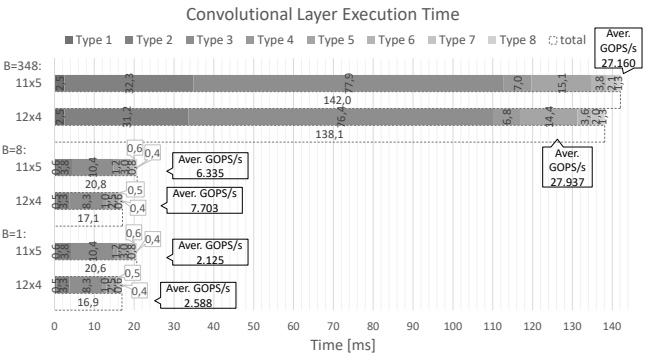


Fig. 12: Execution Time comparison on ECG [22] for NEURAghe 12x4 and 11x5 MAC matrix configurations

Figure 15 shows efficiency and execution time on this benchmark, highlighting contributions of the single layers. Behaviour is similar to the ECG use case on both boards. Executing the processing after each sample ($B = 1$), the efficiency is very limited for all the layers. Increasing B, the accelerator provides much better throughput. The layers executed less efficiently are the early layers, especially the first type, that pays for a significant underutilization of the MAC Matrix, due to the low number of input features. However, its contribution to the overall efficiency is marginal. Layers that show longer execution time are placed at the end of the TCN graph. These layers account for the highest contribution to the overall workload and, as shown in Figure 15b and 16b, are executed quite efficiently when sample batching is used ($B = 144$). Sample batching is more effective on the Zedboard, that integrates a MAC Matrix with less input and output ports, thus requires longer runs to complete convolution, which reduces the impact of data-transfer and accelerator warm-up overheads.

TABLE VII: Convolutional Layer characteristics for Res-TCN [21]

	type 1	type 2	type 3	type 4	type 5	type 6
$inputfeatures$	150	64	64	128	128	256
$outputfeatures$	64	64	128	128	256	256
$Kernel\_size$	8	8	8	8	8	8
$stride$	1	1	2	1	2	1

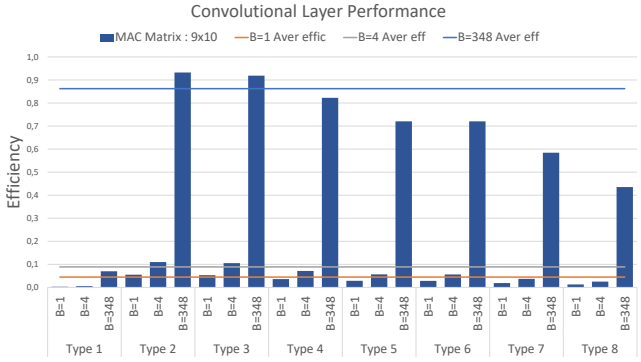


Fig. 13: Efficiency on ECG [22] for NEURAghe 9x10 MAC matrix configuration in ZU3EG

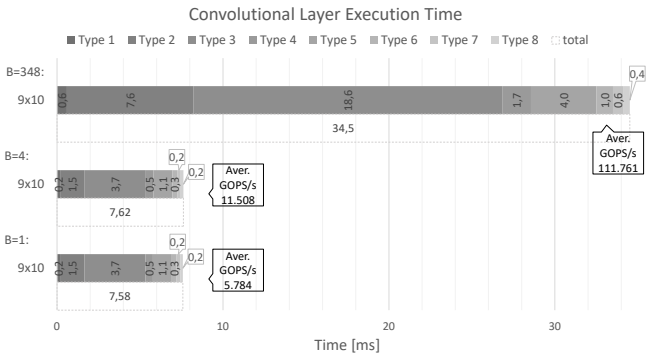
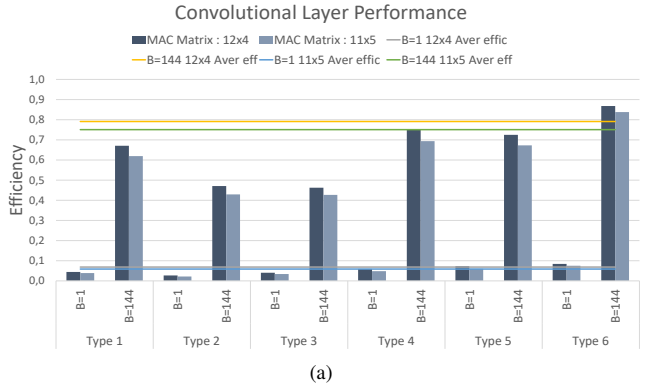
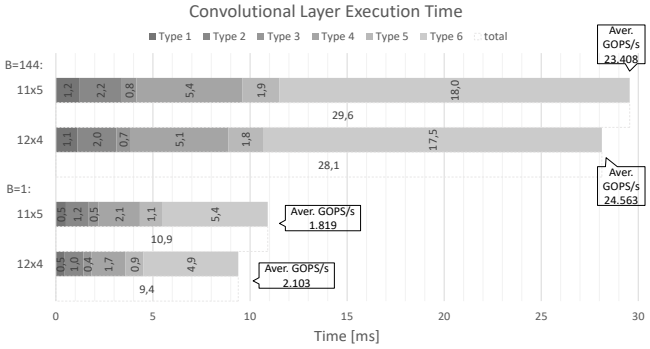


Fig. 14: Execution Time on ECG [22] for NEURAghe 9x10 MAC matrix configuration in ZU3EG



(a)



(b)

Fig. 15: Efficiency trend and Execution Time on Res-TCN [21] for different batch sizes for NEURAghe 12x4 and 11x5 matrix configuration in Z-7020 SoC

C. WN-PNT use-case

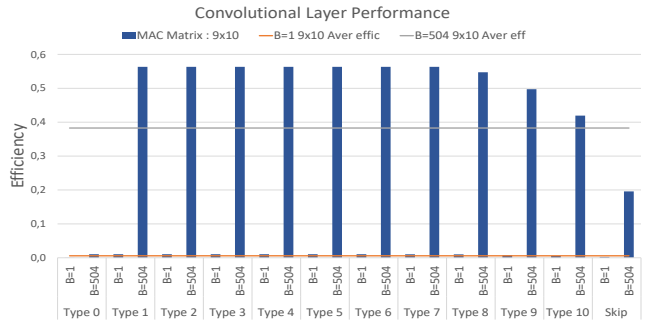
This case considers a network derived from WaveNet (Van Den Oord et al. [19]) for Polyphonic Note Transcription of Time-Domain Audio Signal ([42]). It is made of 20 stacked residual blocks composed by a 1×1 skip connection and a Dilated Convolutional block of 128 channels each. *Filter Size* is 2 and the *dilation factor* grows like 2^k where the residual block index $k \in \{0, 1, 2...9, 0, 1, 2...9\}$.

This case is especially challenging, due to the small size of kernels ($kernel\_size = 2$ in most layers) and because the sample acquisition frequency is $16KHz$, posing hard constraints about real-time execution. Thus we focus on the Ultra-96 board, relying on its higher clock frequency to achieve the required throughput. As may be noticed in Figure 17a, in general, this use case is executed less efficiently on the platform, none of the layers reaches more than 0.56 efficiency. This is because the execution of the actual convolution kernels takes only two cycles and thus hardly overlaps with input data transfers. Batch size can be used to increase the reuse of weights, once they are loaded to the weight memory region, but, at the same time, has an impact on the duration of input and output transfers. To compensate this issue, considering that the total size of the local receptive fields in the network allows for the continuous storage on the on-chip memory for almost all layers, (unless for types 8, 9 and 10), we have slightly modified the scheduling described in Figure 8. In this use-case, we transfer to/from DDR only new output/input samples, while keeping the rest of the local receptive fields in the on-chip memory. As shown in Figure 17b, using $B = 504$ we can process 504 samples in around $19ms$, thus respecting the real-time constraint posed by the use-case.

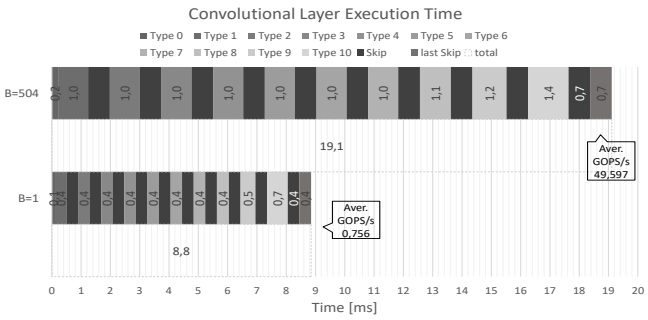
D. Assessment of hardware vs. software speed-up

In order to evaluate the benefits obtained by on-FPGA acceleration, we have compared the execution of the convolution-related load of the three benchmarks on NEURAghe, with the execution on an A53 quad-core ARM processor, using the ARM Compute Library optimized functions [2]. As may be noticed, NEURAghe provides up to 10.8x, 10.7x and 5.7x speed-up on the three reference benchmarks, and considering the power consumption of the two platforms (around $0.9\ W$ for the A53 cores and around $3.3\ W$ for NEURAghe), PL-based acceleration provides improvement, in terms of power efficiency, corresponding to 33,8 GOPS/s/W , 26,3 GOPS/s/W and 15 GOPS/s/W respectively.

[2]Since ARM-CL does not support dilated 1D convolution, for the sake of comparison, we have evaluated execution time on the A53 on analogous convolution layers, without dilation. Considering that in dilated convolutions input samples are not adjacent in memory, the exploitation of the vector processing in the ARM-CL could be suboptimal, thus the execution time reported in Figure 18a for the A53 could be underestimated. Our own software implementation (unoptimized) of a dilated convolution performs one order of magnitude slower than what reported for ARM-CL.

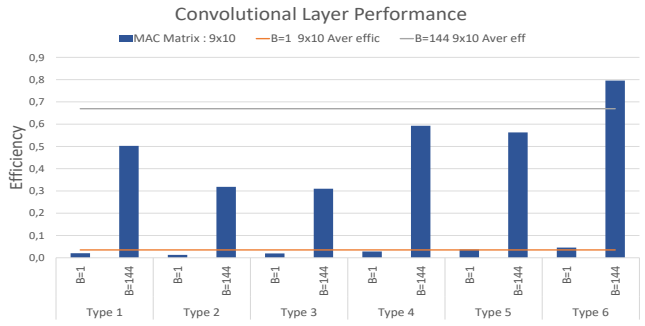


(a)

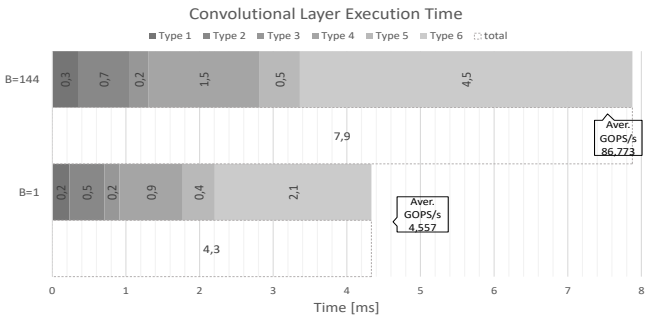


(b)

Fig. 17: Efficiency trend and Execution Time on WN-TCN [42] for different batch sizes for NEURAghe 9x10 matrix configuration in ZU3EG

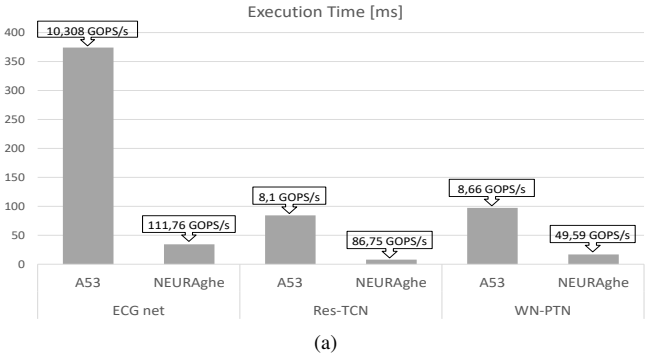


(a)

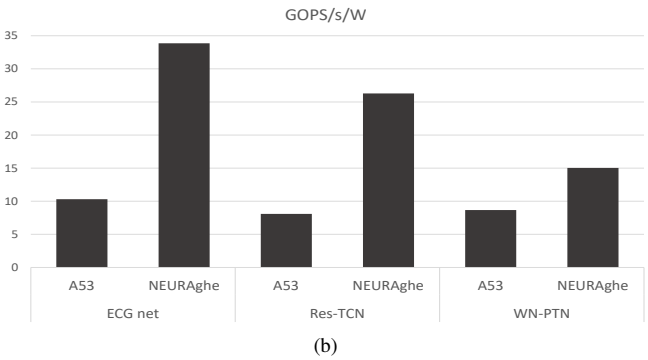


(b)

Fig. 16: Efficiency trend and Execution Time on Res-TCN [21] for different batch sizes for NEURAghe 9x10 matrix configuration in ZU3EG SoC



(a)



(b)

Fig. 18: Execution time and power efficiency comparison between software execution on a Cortex-A53 quad core and NEURAghe (ultra-96).

E. Comparison to other APSoC based accelerators

As mentioned, literature is lacking TCN evaluations on FPGA-based accelerators. The only attempt available is [43], which is strictly customized for a autoregressive TCN and is implemented on a high-end board non usable in the embedded domain. However, when designing the TCN-supporting version of NEURAghe, we have considered compatibility with 2D convolutions for classic CNN acceleration. Moreover, the improved flexibity of the architecture, in terms of kernel size and stride, allows for some improvements with respect to other approaches in the literature targeting low-to-mid All-programmable SoCs. In Table VIII we report comparative results with state-of-the-art on two well known networks for image classification, ResNet-18 and VGG-16. We compare to other accelerator architectures, [44], [45] and [46], that are implemented on the same kind of hardware and use the same 16-bit data precision. On VGG-16, that exposes quite regular kernel sizes and stride values, our work shows comparable performance with respect to the alternatives. It executes convolutions slightly faster than [44] and [45] and around 13% slower than [46]. On ResNet-18, which exposes more variable *kernel sizes* and *strides*, we can reduce a lot of overheads that must be payed by more *static* architectures, executing the whole convolution workload 40% faster than [44].

VIII. Conclusions and Future Work

In this work, we have presented an accelerator architecture supporting Temporal Convolutional Networks, implemented on FPGA-based SoCs. We have presented the accelerator

TABLE VIII: Comparison between this work, previous version ([44]) and other works on ResNet-18 and VGG-16. Xilinx Zynq Z-7020

	This Work	[44]	**This Work**	[44]	[46]	[45]
	ResNet-18		VGG-16			
Xilinx Zynq SoC	**Z-7020**	Z-7020	**Z-7020**	Z-7020	Z-7020	Z-7020
Freq. [MHz]	**120**	120	**120**	120	125	150
GOPS/s	**26,5**	16,1	**42,62**	42.48	48.53	31.38

features serving this specific computing pattern, such as the capability of supporting arbitrary *kernel_size*, *dilation rate* and *stride values* without overhead. We have also shown how data transfers from-to off-chip memory can be managed in TCNs and how performances improve by changing the computational paradigm, going from a *latency constrained* approach to a *batched* approach, that trades latency for throughput. We applied this method to three notable TCNs and using two different SoCs as a target, analyzing throughput, latency and efficiency figures, and the capabilities of the system to respect real-time constraints. Results show that using sample batching, we can achieve efficiency up to 0.96, 0.86 and 0.57 on the three use-cases. In the two use-cases with real-time requirements, an adequate sample batching can be used to achieve sufficient throughput to timely process all input samples. We also compared the execution of the use-cases on a Cortex A53 quad-core, to evaluate the achievable speedup, noticing up to 10x execution time reduction, with an improvement in power efficiency that ranges from 2x to 3.5x depending on the benchmark. Finally, we validated the compatibility of proposed architectural solutions with state-of-the-art CNNs used in the image processing domain, by direct comparison with previous work on APSoC-based accelerator, showing similar performance with respect to CNN-targeting alternatives when regular network topologies are targeted and 40% improvement when targeting more irregular patterns.

REFERENCES

[1] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[2] J. Zhang and N. Verma, "An in-memory-computing dnn achieving 700 tops/w and 6 tops/mm2 in 130-nm cmos," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 358–366, 2019.

[3] Y. Chen, T. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[4] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Hyperdrive: A multi-chip systolically scalable binary-weight cnn inference engine," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 309–322, 2019.

[5] S. Choi, K. Bong, D. Han, and H. Yoo, "Cnnp-v2: A memory-centric architecture for low-power cnn processor on domain-specific mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 598–611, 2019.

[6] M. P. Véstias, "A survey of convolutional neural networks on edge with reconfigurable computing," *Algorithms*, vol. 12, no. 8, p. 154, 2019.

[7] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," *Neural computing and applications*, pp. 1–31, 2018.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[10] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," *CoRR*, vol. abs/1501.02876, 2015, withdrawn. [Online]. Available: http://arxiv.org/abs/1501.02876

[11] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708.

[12] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[13] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 156–165.

[14] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[15] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," *CoRR*, vol. abs/1412.5567, 2014. [Online]. Available: http://arxiv.org/abs/1412.5567

[16] J. E. Weston, "Dialog-based language learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 829–837.

[17] C. D. Santos and B. Zadrozny, "Learning character-level representations for part-of-speech tagging," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1818–1826.

[18] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *arXiv preprint arXiv:1610.10099*, 2016.

[19] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio." *SSW*, vol. 125, 2016.

[20] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," *CoRR*, vol. abs/1612.08083, 2016. [Online]. Available: http://arxiv.org/abs/1612.08083

[21] T. S. Kim and A. Reiter, "Interpretable 3d human action analysis with temporal convolutional networks," *CoRR*, vol. abs/1704.04516, 2017. [Online]. Available: http://arxiv.org/abs/1704.04516

[22] S. Goodfellow, A. Goodwin, D. Eytan, R. Greer, M. Mazwi, and P. Laussen, "Towards understanding ecg rhythm classification using convolutional neural networks and attention mappings," 08 2018.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[25] P. Meloni, A. Capotondi, G. Deriu, M. Brian, F. Conti, D. Rossi, L. Raffo, and L. Benini, "Neuraghe: Exploiting cpu-fpga synergies for efficient and flexible cnn inference acceleration on zynq socs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, Dec. 2018. [Online]. Available: https://doi.org/10.1145/3284357

[26] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[27] X. Yu, Y. Wang, J. Miao, E. Wu, H. Zhang, Y. Meng, B. Zhang, B. Min, D. Chen, and J. Gao, "A data-center fpga acceleration platform for convolutional neural networks," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 151–158.

[28] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–8.

[29] Y. Ma, Y. Cao, S. Vrudhula, and J. s. Seo, "An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2017, pp. 1–8.

[30] S. I. Venieris and C. S. Bouganis, "Latency-driven design for fpga-based convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2017, pp. 1–8.

[31] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: An efficient hardware accelerator for convolutional neural networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.

[32] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 26–35. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847265

[33] A. Prost-Boucle, A. Bourge, F. Petrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on fpga," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2017, pp. 1–7.

[34] Y. Umuroglu, N. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 65–74. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021744

[35] S. Rasoulinezhad, H. Zhou, L. Wang, and P. H. W. Leong, "Pir-dsp: An fpga dsp block architecture for multi-precision deep neural networks," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 35–44.

[36] E. Wang, J. J. Davis, P. Y. Cheung, and G. Constantinides, "Lutnet: Learning fpga configurations for highly efficient neural network inference," *IEEE Transactions on Computers*, 2020.

[37] M. Courbariaux, Y. Bengio, and J. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3105–3113.

[38] "Xilinx Vitis AI development environment," https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.

[39] "Xilinx Deep Learning Processing Unit," https://www.xilinx.com/products/intellectual-property/dpu.html.

[40] "Zynq DPU v3.2," https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_2/pg338-dpu.pdf.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[42] L. Martak, M. Sajgalik, and W. Benesova, "Polyphonic note transcription of time-domain audio signal with deep wavenet architecture," 06 2018, pp. 1–5.

[43] S. Hussain, M. Javaheripi, P. Neekhara, R. Kastner, and F. Koushanfar, "Fastwave: Accelerating autoregressive convolutional neural networks on fpga," *arXiv preprint arXiv:2002.04971*, 2020.

[44] P. Meloni, D. Loi, G. Deriu, M. Carreras, F. Conti, A. Capotondi, and D. Rossi, "Exploring neuraghe: A customizable template for apsoc-based cnn inference at the edge," *IEEE Embedded Systems Letters*, vol. PP, pp. 1–1, 10 2019.

[45] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[46] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for fpga-based convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.