A Survey on CNN and RNN Implementations

Javier Hoffmann, Osvaldo Navarro, Florian Kästner, Benedikt Janßen, Michael Hübner
Chair for Embedded Systems of Information Technology
Ruhr-Universität Bochum
Bochum, Germany
Email: {Javier.Hoffmann;Osvaldo.NavarroGuzman;Florian.Kaestner;Benedikt.Janssen;Michael.Huebner}@rub.de

***Abstract*—Deep Neural Networks (DNNs) are widely used for complex applications, such as image and voice processing. Two varieties of DNNs, namely Convolutional Neuronal Networks (CNNs) and Recurrent Neuronal Networks (RNNs), are particularly popular regarding recent success for industrial applications. While CNNs are typically used for computer vision applications like object recognition, RNNs are well suited for time variant problems due to their recursive structure. Even though CNNs and RNNs belong to the family of DNNs, their implementation shows substantial differences. Besides more common Central Processing Unit (CPU) and Graphic processing Unit (GPU) implementations, Field Programmable Gate Array (FPGA) implementations offer great potential. Recent evaluations have shown significant benefits of FPGA implementations of DNNs over CPUs and GPUs. In this paper, we compare current FPGA implementations of CNNs and RNNs and analyze their optimizations. With this, we provide insights regarding the specific benefits and drawbacks of recent FPGA implementations of DNNs.**

Keywords–deep learning, convolutional, recurrent, neural network.

I. INTRODUCTION

Deep learning is a powerful method for supervised learning without the need of feature engineering. The resulting Deep Neural Networks (DNN) can represent functions of high complexity due to the size of the network depending on the number of hidden layers and neurons or units inside each layer [1]. Two varieties of DNNs, Convolutional Neural Networks (CNNs) and Recursive Neural Networks (RNNs), have especially shown great potential to real-life applications. While CNNs are typically used for imagery classification [2][3], RNNs are suitable for time-variant problems like speech recognition [4] due to their recursive structure.

A. Convolutional Neural Networks

CNNs are a type of feed-forward artificial neural network [1]. Figure 1 shows a diagram of a simple CNN. CNNs usually consists of the following components: convolution, evaluation of a non-linear activation function, pooling or sub sampling and classification or regression. The convolution component extracts features from the input image with a set of learnable filters or kernels called feature maps. The convolution computation is done through a dot product between the entries of the kernel and the input section of same size across the entire input frame. The output of the dot-product is forwarded to an activation function, typically $sigmoid()$, $ReLU()$ or $tanh()$, which increases the nonlinear properties of the CNN. Then, the pooling component reduces the dimensionality of each feature map and keeps the most important information for the classification component. In modern CNNs, like ImageNet [5], these three stages alternate several times. At the next stage of the feed-forward path, one or more fully-connected layers are applied to extract the classification or regression information with the help of logistic or linear regression. The training of CNNs is done via gradient-based backpropagation algorithm.

B. Recurrent Neural Networks

RNNs extend the concept of conventional feed-forward neural networks by having a hidden recurrent state. The activation of the hidden recurrent state depends on the previous activations [6]. Thus, unlike conventional neuronal networks, they can include timing information in the processing, which is important for instance for speech recognition [7].

One of the first approaches in the direction of today's RNNs was done by Jain et al. They developed a partially RNN to learn character strings [8]. The feedback connection enables RNNs to represent previous inputs as activation [9]. According to Jain et al. [8] the architecture of RNN can be anything between a fully interconnected network and a partially interconnected network. In fully interconnected networks, every neuron is connected to every other neuron, as well as itself via feedback connections, allowing to transfer states during-run time. This is depicted in Fig. 2, where the unfolded structure at the bottom shows the behavior of a self-feedback connection. However, in contrast to Fig. 2, for a fully interconnected RNN, there are no distinct input and output layers.

In the 90s, there have been difficulties to train RNNs for applications, whose data includes dependencies over long temporal intervals. The first contribution to a network to overcome the issue of learning long-term dependencies was made by Hochreiter and Schmidhuber [9]. They proposed a new recurrent network architecture, together with a gradient-based learning algorithm, called Long Short-Term Memory networks (LSTM). These networks can be categorized as a sub-group of RNNs. The network architecture LSTM cells is depicted in Fig. 3.

C. CNN and RNN Implementations

There have been several approaches to accelerate the training and the feed-forward path of CNNs and RNNs. Within the scope of this paper, we focus on the forward-path implementation on different platforms: Aplication Specific Integrated (ASIC),Fiel Programmable Gate Array (FPGA) and Graphic Processing Unit (GPU). These platforms distinguish in terms of performance, flexibility and energy consumption.

The use of CNNs and GPUs as accelerators has turn common, achieving significant speedups and better scalability in comparison with a general-purpose processor [10]. The possibility to train DNNs with the help of GPUs is one of the main reasons of their recent success, due to the high-performance speedup caused by the utilization of massive

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-549-4

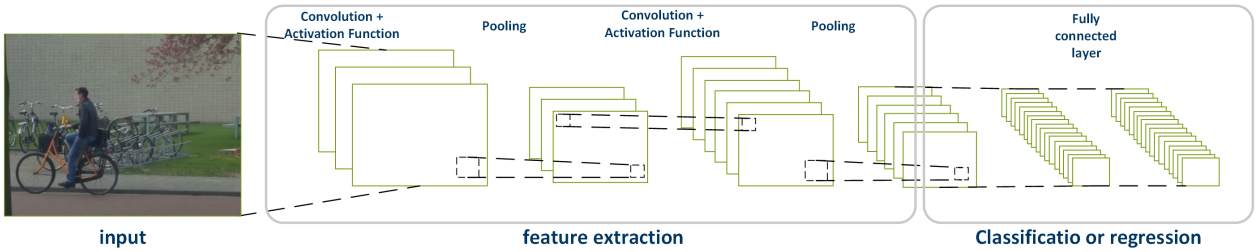

Figure 1. Common processing flow of a convolutional neural network [1].

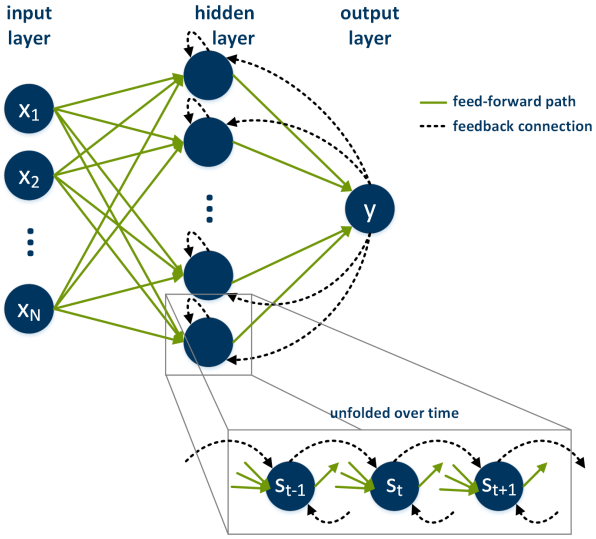

Figure 2. Principle of RNNs, layers are connected to their predecessors via feedback connections.

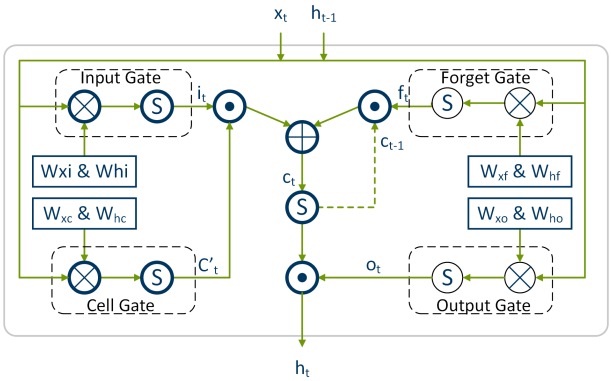

Figure 3. LSTM architecture with four gated units. The S denotes the Sigmoid function [7].

parallelism and batch processing. However, also ASIC and FPGA implementations can be beneficial. The acceleration of the feed-forward or inference path of DNNs on FPGAs offers benefits like a better utilization of fine-grained or heterogenous parallelism, as well as a higher energy-efficiency and flexibility. These advantages are especially interesting with respect to the application of DNNs on embedded devices. In this paper, we focus on analyzing different hardware architectures, mainly FPGA implementations, for accelerating CNNs and RNNs. The structure of our paper is as follows: Section I briefly introduces our paper with a description of CNNs, RNNs and their implementation platforms. Section II presents an overview of the recent implementations of CNNs. Next, Section III describes an analysis on the implementation of CNNs on FPGAs boards. Similarly, Section IV presents an analysis of recent implementations of RNNs on FPGA boards. Finally, Section V presents the concluding remarks of our work.

II. Overview of Convolutional Neural Networks Implementations

Within this section, we give an overview about multiple CNN implementations. The following Section III discusses FPGA implementations in more detail.

A. ASIC-based Implementations

ASIC-based approaches offer great performance and low power capabilities; however, they lack flexibility and have a larger design flow process. The approaches based on this platform are highly customized for an application. Under this category, Chen et al. [11] introduced $Eyeriss$, accelerator for deep CNNs that aims for energy efficiency. The accelerator consists of a Static Random-Access Memory (SRAM) buffer that stores input image data, filter weights and partial sums to allow fast reuse of loaded data. This data is streamed to a spatial array of 14x12 Processing Engines (PE), which compute inner products between the image and filter weights. Each PE consists of a pipeline of three stages that computes the inner product of the input image and filter weights of a single row of the filter. The PE also contains local scratch pads that allow temporal reuse of the input image and filter weights to save energy. There is also another scratch pad in charge of storing partial sums generated for different images, channels or filters, also with the aim of reusing data and saving energy. To optimize data movement, the authors also propose a set of input Network on Chips (NoC), for filter, image and partial sum values, where a single buffer read is used by multiple PEs. This approach was implemented in a $65nm$ CMOS, achieving a frame rate $44.8fps$ and a core frequency of $250MHz$.

Korekado et al. [12] propose an analog-digital hybrid architecture for CNNs for image recognition applications, focusing on high performance at low power consumption. The architecture consists of a pulse-width modulation circuit to compute the most common operations of a CNN and a digital memory to store intermediate results, making use of a time-sharing technique to execute the operations required by all

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-549-4

the connections of the network with the restricted number of processing circuits available in the chip. This architecture was implemented in a $0.35\mu m$ CMOS. The paper reports an execution time of $5ms$ for a network with 81 neurons and 1620 synapses, an operation performance of 2 Giga Operations Per Second (GOPS) and a power consumption of $20mW$ for the Pules Width Modulation (PWM) neuron circuits and $190mW$ for the digital circuit block.

Andri et al. [13] present $Yodann$, an ASIC architecture for ultra-low power binary-weight CNN acceleration. In this work, a binary-weight CNN is chosen for implementation because limiting a CNN's weights to only two values (+1/-1) avoids the need of expensive operations such as multiplications, which can be replaced by simpler complement operations and multiplexers, thus reducing weight storage requirements. This also has the advantage of reducing I/O operations. Moreover, this approach implements also a latch-based standard cell memory (SCM) architecture with clock-gating, which provide better voltage scalability and energy efficiency than SRAMs, at the cost of a higher area consumption. The architecture was implemented using UMC $65nm$ standard cells using a voltage range of $0.6V - 1.2V$. The article reports a maximum frequency of $480MHz$ at $1.2V$ and $27.5MHz$ at $0.6V$ and an area of 1.3 Million Gate Equivalent (MGE).

B. GPU-based Implementations

Recently, GPUs have been used as an implementation platform for highly parallelizable algorithms. While they offer more flexibility than the ASICs, they are more limited than the FPGAs in terms of customization capabilities. Ciresan et al. [14] presents a parameterizable GPU implementation for CNN variants. This implementation enables the user to configure several structural CNN parameters such as: input image size, number of hidden layers, number of maps per layer, kernel sizes, skipping factors and connection tables. Regarding the architecture, this implementation consists of three main CUDA kernels: Forward Propagation (FP), Backward Propagation (BP) and Adjusting Weights (AW). In the FP and BP kernels, each map of a CNN that has to be computed is assigned a different thread block to allow parallelization. The AW kernel is implemented as a 1D grid, with one block located between each connection between two maps, where each block is a 2D grid, with a corresponding thread for every kernel weight. This approach was evaluated with a set of benchmarks for different applications in the domain of object recognition: digit recognition (MNIST), 3D object recognition (NORB [15]) and natural images (CIFAR10 [16]). The authors report a speedup of 10 to 60 in comparison with a compiler optimized CPU. Moreover, this approach has been used to implement deep neural networks for traffic sign classification, which achieved a better-than-human recognition rate [17] and for automatic segmentation of neuronal structures from stacks of Electron Microscopy (EM) images [18]. Furthermore, in order to ease the implementation of deep learning approaches on GPUs, some frameworks and libraries have been proposed as well.

C. FPGA-based Implementations

FPGA-based approaches take advantage of the flexibility and rapid-prototyping capabilities that this platform offers. However, FPGAs are still more power hungry than ASICs. A highly cited deep learning approach implemented on an FPGA is the one of Farabet et al. [19], a hardware architecture to accelerate CNNs for synthetic vision systems. The architecture consists of a control unit, several parallel ALUs/Streaming operators and a memory interface streaming engine. The ALUs implement operators required by bio-inspired CNNs, such as 2D convolver, dot products between a vector and multiple 2D planes, spatial pooling, arbitrary non-linear mappings, element-wise multiplication of a vector by another one. This approach was implemented in a CPU, an FPGA (a Xilinx Virtex-4 SX35) and was also simulated for an ASIC.

Chakradhar et al. [20] propose a co-processor configurable architecture to accelerate CNNs. The authors show evidence of how different workloads and different number of inputs and outputs need different network architectures to yield optimal performance. Thus, they propose an architecture that analyses workloads, can configure the number of convolvers in a computational element and also, the number of computational elements in the whole network to match the types of parallelism of every workload. The architecture was implemented on a Virtex 5 SX240T FPGA and was compared against the best fixed architecture for a set of benchmark applications.

Farabet et al. [21] introduce $NeuFlow$, which is a reconfigurable dataflow processor for vision algorithms. The architecture of this processor consists of a 2D grid of $ProcessingTiles$ (PT). Each PT can be configured to carry out operations on a stream of data, from unary operations to complex nested operations. The grid is connected to a $SmartDMA$, which is connected to an Off-chip memory. The $SmartDMA$ allows the control unit to configure the grid and the DMA ports for each operation and to carry the data from the grid to the off-chip memory in parallel. To configure the processor for a particular algorithm, a compiler called $LuaFlow$ was developed. This compiler takes a tree or a graph structure that represents the algorithm and parses it to determine the parallelism of the algorithm that the processor can take advantage of. This approach was implemented on a Xilinx Virtex 6 ML605 FPGA board at $200MHz$ using 10x10 grids. At its peak performance, the design achieved 80 billion connections per second. Furthermore, a CNN for street scene parsing was implemented on this platform, achieving a performance of $12fps$.

It is hard, if not impossible, to generate a fair comparison among all the described approaches, because they were either designed for different applications with different constraints and objectives or they were implemented using different devices with different configurations. Table I shows a summary of these values, categorized by the three main implementation platforms. Furthermore, the following section will stablish a comparison between approaches from the year 2016 against that of 2011.

III. Analysis of CNN Implementation FPGAs

The selected baseline FPGU implementation of Farabet et al. is presented in [21] and [24]. The modules are implemented on the processing tiles according to the operators contained in each of them, for instance, those with multiply-accumulate operators can be used to execute a 2D convolution (*Convolutional module*) while those with the *tanh operator* are dedicated to the non-linear module.

For the pooling layer, it is possible to use the same processing tiles mentioned before for the convolution module.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-549-4

TABLE I. COMPARISON OF CNN IMPLEMENTATION APPROACHES

Platform	Approach	Device	Core Frequency	Power	FPS	Performance	Area
ASIC	Eyeriss [11]	65nm CMOS	100 - 250MHz	278 mW at 1V	44.8	16.8 - 42 GOPS	1852 kGates
	[12]	0.36μm CMOS		20mW PWM circuit + 190 mW digital core		2 GOPS	
	Yodann [13]	65nm UMC	27.5MHz-480MHz	895μW	11 - 16.8	1.5 TOPS	1.3 MGE
	[19]	ASIC Simulation	400MHz	1W	>30		
FPGA	[19]	Xilinx Virtex-4 SX35	200MHz	15W	> 30		
	[20]	Virtex 5 SX240T	120MHz	<14W	25 - 30		
	NeuFlow[21]	Virtex 6 ML605	200MHz	10W	12	160 GOPS	
	[22]	Altera Stratix	130MHz		1600	409.622 GOPS	
	[23]	Xilinx Kintex-7		<4W		247 GOPS	
GPU	[14]	1 CPU: Core i7-920, 4 GPU: 2 x GTX 480, 2 x GTX 580	2.66 GHz (CPU)			10-60 speedup vs CPU	

Here is also stated that it is advantageous to use a 16Bit format instead of a 32, then, even though a loss of precision is caused, this is not a relevant loss. The results on each of the processing tiles is also pipelined to obtain a result for each clock cycle.

This approach was tested on the datasets NORB[**?**], MNISt[25] and UMASS[26] obtaining correct classification between 85% and 98% depending on the dataset.

The next step is to present 2 papers published 5 years later, in 2016, claiming to have encountered a novel and performant implementation.

On the one hand, Li et al. [22] focus on High-speed image classification, mentioning that the implementation presented can be useful if implemented in a rear-end collision avoidance system. On the other hand, Dundar et al. [23] warrant that the approach supports different configurations of deep CNNs, disregarding layers depth, filter quantity or others, and it is stated that the a perfect scenario for its application is the generic image processing. The former utilizes an Altera Stratix V 5SGSMD5K2F40C2 FPGA while the latter a Xilinx Kintex-7 XC7K325T.

On the filter layer, [22] utilizes 3D convolvers for each filter. These are composed of multiple 2D convolvers that calculate one channel of the filter. The 2D convolvers contain the so called *Processing Elements (PE)* connected under a First-In First-Out (FIFO) queue. The PE are, at their time, composed f multipliers and adders, like the processing tiles presented before, plus a register of one word length.

For [23] the convolution on this layer is done with trained filters to obtain remarking features from the images. The weights are represented as 4D filters.

For the non-linear module, both researches employ Rectified-Linear Units (ReLU) since it is said to be more suitable to implement in Hardware.

There are, again, some differences in the pooling layer, where [22] considers an approach closer to the one presented in [24] where the pooling and filter bank layer are executed in a similar fashion, even if with some minor modifications. These differences are, for example, that the PEs compare two different values and return only the higher value, reducing the output and energy consumption, this is not present in [19]. For [23] however, the approach is similar. The input images of the layer are divided in smaller squares, windows, and in a comparable manner to the previously explained, only the maximum value of the resulting set is to be outputted, this is a mode of non-linear down-sampling, and the factor of down-sampling is equal to the size of the windows.

As a novel idea for improving the performance of the system, [22] presents the *Global Summation*, designed to minimize resource consumption on the latest layers in favor of the overall performance. In contrast, [23] presents several techniques to improve the control method of the process, among several improvements. For instance, it is important to mention the *input reuse* and the *concatenation of data*. The Input reuse, consists of extending the information to convolution modules even though if the amount of available ports is smaller than that of free convolution modules. Those convolution modules that have access to the port can share the information to those who are idle. The concatenation of data profits of the Q8.8 format. The Q8.8 format uses 16 bits in a 32 stream, so that 2 different data words can be transmitted in a unique stream

Finally, these implementations were tested on datasets, in the case of [22], this was done over two different datasets, the first consisting of several images from the datasets CIFAR[16], CBCL[27] and Caltech Cars [28]. The second one consisted of edited images from Caltech Cars and CBCL car dataset. They achieved 94.5% in the first case and 98.4% in the second. The system is said to be performant at 1,600 frames per second and according to estimations of the authors, they obtain 409,62 GOPS.

[23] on the contrary, was tested on an own dataset utilizing videos and sets of images from these videos. Their work was compared to a GPU NVIDIA K40, a Zynq ARM Cortex A9 at 667MHz, an Intel Core i7 at 2.5GHz and a naïve implementation of a hardware accelerator. Regarding performance per second, the GPU turned to be victorious with a huge advantage over the others, however, when calculating the performance per Watt, the implementation presented was more performant than the others. The authors mentioned to have a peak in the performance at 247 GOPS.

Both approaches present FPGAs as a tool for accelerating deep CNNs and both belong to the newest researches on this field. The improvement on the performance of each implementation is notable when compared to an implementation 5 years older. However, the core idea of the accelerator presented in 2011 is still present in these two newer papers.

When comparing the three CNN modules, the approaches shared similarities, as for example, the use of ReLU for the activation layer, instead a sigmoid function or a tanh operation.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-549-4

Another important similarity is to be found in the pooling layer, where only the higher value of the resulting set is to be outputted.

The difference between these approaches lies in the implementation of the convolutional layer, where the representation of the filters is very different for each of them.

Another small but very important similarity is the usage of the number format Q8.8, of 16 bits. [23] goes further and takes advantage of this by sending two different words in a single bitstream. Important is also to notice that [22] used well known datasets, CIFAR, Caltech cars, to test this approach, while [23] used their own dataset. When comparing the value of GOPS, [22] has a value that almost doubles the highest peak of [23]. However this value is obtained from a simulation of the model while the value on [23] is an output design.

IV. Analysis of RNN Implementations on FPGAs

In general, two possibilities exist to increase the overall performance of RNNs regarding the inference phase on FPGAs. On one hand the computations which must be done while passing the input through the network can be optimized. This includes adjusting the level of parallelism due to the hardware constraints, the choice of hardware accelerator module and approximations of computation, in case of an acceptable minimal loss of precision. On the other hand, the data communication, which includes weight parameters and inputs, between the FPGA and an external memory, like Dynamic Random Access Memory (DRAM), can be optimized to improve the throughput. This is necessary because typically the FPGA on-chip-memory, like Block RAM (BRAM) or distributed memory, is too small to store all parameters of real-life RNNs. Thus, the developer of FPGA accelerators for RNNs have to consider both optimization paths. Moreover, to increase the overall performance both optimizations paths must collaborate to avoid bottlenecks.

Guan et al. [7] faced both optimization paths. After profiling a typical RNN inference structure they found out that the main bottleneck is caused by the computations, which mainly include floating point multiplication and addition, inside each LSTM gate. Therefore, they separate the computation scheme (Fig.3) into four LSTM gate modules, which output the input, forget, cell and output gate vector executed in parallel, and a LSTM Functional Logic, which includes the following computations. Inside each gate the multiplications of input and parameter vector are also done in parallel and summed up through an addition tree. The activation function at the end of each gate module and the LSTM functional logic are replaced with a piecewise linear approximation of $sigmoid$ and $tanh$, which was originally introduced by Amin et al. [29]. As a result, resource and performance inefficient computations are substituted with simpler and more hardware efficient addition and shifting operations with a minimal loss of accuracy. Furthermore, two input and output ping-pong buffer groups are improving the communication performance. The parameters are stored in an external DRAM and reshaped offline so that the data access can be done in a sequential manner. Moreover, a data dispatcher is used to interface the DRAM with full burst size of the AXI4 bus. The Accelerator is designed with a High-Level Synthesis tool (HLS), Vivado HLS from Xilinx [30] and the system is implemented on a XILINX VC707 device with a Virtex7 FPGA chip. The evaluation shows that the system achieves a maximum performance of 7.26 GFLOP/S.

A different architecture is proposed by Chang et al. [31]. Like in the previous architecture, they use two different hardware accelerator modules, to separate gating and LSTM-logic computations. Another similarity is the simplification of the nonlinear activation function through piecewise linear approximation. In contrast to the previous design Chang et al. [31] use fixed point 16-bit operations for MAC (Multiply Accumulate) operations resulting in 32 bit values for further operations. While fixed-point computations are far more efficient for hardware implementations, a loss of accuracy has to be considered what was denoted as a maximum of 7.1. Furthermore, the gating computations are separated into two sequential steps. The input and cell gate computations are done in parallel as well as the output and forget gate computations. As a result, the output of the LSTM module is provided after 3 sequential steps. Thus, the course grained parallelism is not fully exploded. For communication optimization, the authors of [31] use a combination of memory mapping and streaming interface. Therefore, four Direct Memory Access (DMA) are used to access the external DRAM and reshape the data to be forwarded through 8 AXI4-Stream, which activity depends on the current routing, and buffered with FIFOs. In comparison to the communication architecture presented by Guan et al. [7] the methodology is equal but the differences in implementation details mainly arise because of the different course grained parallelism of the LSTM accelerator. The design is implemented on a Zedboard with Zynq 7020 FPGA from Xilinx. The hardware was tested with a two-layer LSTM including 128 hidden units running with a frequency of 142MHz. The performance was outlined to 264.4 million operations per second.

In contrast to [31] and [7] the authors of [32] use a retrained based method to reduce the word-length of the weights. As a result, they achieve a quantification to 6 bits per weight. Due to this fact, for the desired speech recognition application all weight parameters can be stored on the on-chip-memory (BRAM) on the XC7Z045 FGPA device from Xilinx without loss of precision differences of retraining and hardware implemented inference computation. Without the requirement to optimize the communication from an external DRAM the focus of this work is the accelerator module optimization. Although the proposed LSTM accelerator module is separated into two different modules the parallelism and task granularity differ from those proposed in [7] and [31]. All matrix-vector multiplications are computed inside one Processing Element (PE) array, which consist of 512 PEs. The remaining computations including evaluating activation functions are done using an additional block called Extra Processing Unit (EPU). The outputs of the PE array are buffered and forwarded to the LSTM EPU. As mentioned above, the design benefits from a high level of fine-grained parallelism. However, the architecture is not comparable to the other design as no communication bottleneck has to be handled and the authors focus on the optimization of the whole speech recognition algorithm while considering real-time constraints for the desired application.

Besides, Ferreira et al. [33] follow a modular extensible architecture. They assume a similar reduction of communication complexity as assumed in [32]. In contrast to [32] the word-length of the weights are 18 bits, which leads

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-549-4

to a high utilization of the Digital Signal Processor (DSP) DSP48E1 slices of the FPGA, and they are stored in a Look-up RAM (LUTRAM). On the one hand, this distributed memory represents the fastest way of accessing the weights due to the closeness to the accelerator and the unlimited simultaneous port access to each LUTRAM array. In contrast, BRAM supports a maximum of two port access. On the other hand, this method consumes a high amount of resources especially when the implemented network has many layers and LSTM-tiles. The architecture does not explore full parallelism in a coarse-grained manner. The matrix-vector multiplications are done in parallel for all four gates. Due to the negligible amount of additional clock cycles needed for elementwise multiplication and activation function evaluation of $tanh()$ and $sigmoid()$, which are approximated with ridge (polynomial) regression. Each of these computations own one hardware module. The gate outputs are forwarded to a multiplexer and further routed to the desired hardware module. The design was implemented on a Xilinx XC 7Z020SoC device with different amounts of neurons per layer. The network size is not allowed to extend 31 neurons per layer due to the limited number of DSP-slices on the device. The frequency of the hardware accelerator was adjusted to maximum with respect to the layer size. Thus, the design is capable of achieving 4534.8 million operation per second, which is 17 times more than the performance reached in [31].

V. Conclusion

As outlined in the previous sections, there are many possibilities to accelerate CNNs or RNNs with the help of hardware modules. Each of these hardware implementations has a significant speedup in comparison to a CPU implementation. From our perspective, there are several reasons making a fair comparison of all hardware architectures not possible. One of that reasons is that the size of the networks mostly depends on the application as well as the real-time constraints. Thus, a good hardware implementation reduces energy and resource consumption without validating real-time constraints.

Regarding the evolution of the implementation of CNNs on a FPGA over five years, it is remarkable that the core framework has not changed abruptly in this time lapse, where the researches utilizes different FPGUs platforms. Nevertheless, the most important object of study are the modules. These different approaches apply different metrics to stablish the performance success, i.e. some use GOPS, FLOPS or Fps, while other focus on the energy consumed, making a direct comparison among them a very hard task. It would be ideal if a baseline reference statistics could be defined, so that future works compare themselves to this baseline. As Table I shows, even though the performance between FPGA and ASIC approaches are comparable, the reported power consumption values of the latter are lower. This is usually the tradeoff between FPGA and ASIC implementations, i.e. higher flexibility but higher power consumption. However, this can be improved by simplifying the DNN architecture with little loss of accuracy. One nice example of this are BNNs [13], which achieve a competitive accuracy and greatly decease power consumption. This simplification could be also coupled with the reconfiguration capabilities of FPGAs, such that this tradeoff of accuracy and power consumption can be adapted to the requirements of the application [20].

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.

[2] W. Zhao, S. Du, and W. J. Emery, "Object-based convolutional neural network for high-resolution imagery classification," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. PP, no. 99, 2017, pp. 1–11.

[3] H. J. Jeong, M. J. Lee, and Y. G. Ha, "Integrated learning system for object recognition from images based on convolutional neural network," in 2016 International Conference on Computational Science and Computational Intelligence (CSCI), Dec 2016, pp. 824–828.

[4] T. He and J. Droppo, "Exploiting lstm structure in deep neural networks for speech recognition," in 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2016, pp. 5445–5449.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105, last accessed March 27th, 2017. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[6] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," CoRR, vol. abs/1412.3555, 2014, last accessed March 28th, 2017. [Online]. Available: http://arxiv.org/abs/1412.3555

[7] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks," in 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), Jan 2017, pp. 629–634.

[8] L. C. Jain and L. R. Medsker, Recurrent Neural Networks: Design and Applications. CRC Press, Inc., 2001.

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

[10] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on. IEEE, 2010, pp. 317–324.

[11] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, 2017, pp. 127–138.

[12] K. Korekado, T. Morie, O. Nomura, H. Ando, T. Nakano, M. Matsugu, and A. Iwata, "A convolutional neural network vlsi for image recognition using merged/mixed analog-digital architecture," in Knowledge-Based Intelligent Information and Engineering Systems. Springer, 2003, pp. 169–176.

[13] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An architecture for ultra-low power binary-weight cnn acceleration," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. PP, no. 99, 2017, pp. 1–1.

[14] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22, no. 1. Barcelona, Spain, 2011, p. 1237.

[15] "NORB Dataset," http://www.cs.nyu.edu/ ylclab/data/norb-v1.0/, accessed: 2017-04-15.

[16] "CIFAR Dataset," https://www.cs.toronto.edu/ kriz/cifar.html, accessed: 2017-04-15.

[17] D. CireşAn, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," Neural Networks, vol. 32, 2012, pp. 333–338.

[18] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in Advances in neural information processing systems, 2012, pp. 2843–2851.

[19] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for syn-

thetic vision systems," in Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on. IEEE, 2010, pp. 257–260.

[20] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in ACM SIGARCH Computer Architecture News, vol. 38, no. 3. ACM, 2010, pp. 247–257.

[21] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on. IEEE, 2011, pp. 109–116.

[22] N. Li, S. Takaki, Y. Tomiokay, and H. Kitazawa, "A multistage dataflow implementation of a deep convolutional neural network based on fpga for high-speed object recognition," in 2016 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI), March 2016, pp. 165–168.

[23] A. Dundar, J. Jin, B. Martini, and E. Culurciello, "Embedded streaming deep neural networks accelerator with applications," IEEE Transactions on Neural Networks and Learning Systems, vol. PP, no. 99, 2016, pp. 1–12.

[24] C. Farabet, Y. LeCun, K. Kavukcuoglu, and E. Culurciello, "Large-scale fpga-based convolutional networks," Scaling up Machine Learning: Parallel and Distributed Approaches, pp. 399–419.

[25] "MNIST Dataset," http://yann.lecun.com/exdb/mnist/, accessed: 2017-04-15.

[26] "UMAss Dataset," https://www.umass.edu/statdata/statdata/index.html, accessed: 2017-04-15.

[27] "CBCL Dataset," http://poggio-lab.mit.edu/, accessed: 2017-04-15.

[28] "CALTECH Dataset," http://www.vision.caltech.edu/archive.html, accessed: 2017-04-15.

[29] H. Amin, K. M. Curtis, and B. R. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of a neural network," IEE Proceedings - Circuits, Devices and Systems, vol. 144, no. 6, Dec 1997, pp. 313–317.

[30] "Vivado hls," https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html, accessed: 2017-04-15.

[31] E. C. Andre Xian Ming Chang, Berin Martini, "Recurrent neural networks hardware implementation on fpga," in arXiv preprint arXiv:1511.05552, 2015.

[32] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "Fpga-based low-power speech recognition with recurrent neural networks," in 2016 IEEE International Workshop on Signal Processing Systems (SiPS), Oct 2016, pp. 230–235.

[33] J. C. Ferreira and J. Fonseca, "An fpga implementation of a long short-term memory neural network," in 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Nov 2016, pp. 1–8.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-549-4