2023 XXIX International Conference on Information, Communication and Automation Technologies (ICAT) | 979-8-3503-9983-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICAT57854.2023.10171329

SqueezeJet-3: An HLS-based Accelerator for Edge CNN Applications on SoC FPGAs

Panagiotis Mousouliotis, Nikolaos Tampouratzis, Ioannis Papaefstathiou
School of Electrical and Computer Engineering
University of Thessaloniki
Thessaloniki, Greece
{pmousoul,ntampouratzis,ygp}@ece.auth.gr

***Abstract*—Most FPGA-based Convolutional Neural Network (CNN) hardware accelerators target the datacenter rather than edge processing units. To further fill this gap, this work presents SqueezeJet-3 and the corresponding design flow of a novel FPGA-based embedded system, consisting of software and hardware for accelerating edge CNN inference. SqueezeJet-3 is optimized for accelerating small ImageNet class CNNs, such as SqueezeNet v1.1 and ZynqNet, on low-end low-cost SoC FPGA devices. SqueezeJet-3 is evaluated against the DietChai accelerator, which is part of Xilinx's ChaiDNN v2 framework, in terms of performance, resource utilization, power, and accuracy; the results demonstrate that for the acceleration of SqueezeNet v1.1, SqueezeJet-3 is better than DietChai in all categories. Our evaluation results also show that, by using the presented design framework, a developer can implement FPGA accelerators for larger CNNs, such as the VGG16, with similar performance to the accelerators designed by Angel-Eye and fpgaConvNet frameworks which are optimized for VGG16-like CNN networks.**

***Index Terms*—Algorithm-to-HLS Workflow, High-Level Synthesis, FPGA CNN Accelerator, Deep Learning Application, Mobile Embedded Systems**

I. Introduction

Today's FPGA CNN accelerators and the related frameworks [1] usually target large CNNs and high-capacity FPGA devices which are unsuitable for edge applications. Edge CNN applications, for example those used in drones, require reduced latency, power, and cost. Although CNN research trends move towards developing efficient architectures of small and accurate CNNs [2], the research on FPGA-based CNN accelerators mainly focuses on high-end cloud/server infrastructures. Towards this end, we present the architecture, the implementation, and the corresponding design-flow of SqueezeJet-3, an FPGA-based CNN accelerator which can be used to accelerate small CNNs such as SqueezeNet v1.1 [3] and ZynqNet [4]. The contributions of our work can be summarized in the following:

- the implementation of the Ristretto dynamic fixed-point quantization scheme and its verification for utilization in FPGA accelerators
- the presentation of SqueezeJet-3, an accelerator which can be used to accelerate CNNs for edge applications
- the acceleration of the SqueezeNet v1.1 CNN using Xilinx's DietChai accelerator
- the comparison between SqueezeJet-3 and Xilinx'x DietChai, Angel-Eye, and fpgaConvNet accelerators in terms of latency, resource utilization, power, and accuracy

The rest of this work is organized as follows: section II presents the related work, section III describes the SqueezeJet-3 accelerator design flow as part of the CNN-Grinder framework[1], section IV provides information related to the architecture and the operation of the SqueezeJet-3 accelerator, section V presents the experiments used for the evaluation of our accelerator, section VI reports the results, and finally section VII concludes the paper.

II. Related Work

In this section we present the main FPGA-based systems used for accelerating edge CNN applications. The related work can be categorized into: (1) designs that accelerate small but accurate CNNs suitable for edge applications which fit in low-end low-cost FPGA devices and have low power consumption, (2) accelerators which are latency optimized (e.g. use of a batch size equal to 1), and (3) systems that utilize CNN compression methods (e.g. quantization).

ZynqNet [4] comprises mainly of a design method for implementing FPGA-based CNN accelerators. ZynqNet is both a CNN architecture suitable for the Zynq SoC FPGAs and at the same time an HLS design flow. The authors of ZynqNet develop a CNN architecture by replacing in the SqueezeNet CNN architecture [3] the combination of the convolutional and max-pooling layers with convolutional layers with increased stride [5]. Other FPGA-based CNN accelerators are the SqueezeNext, the ShiftNet, and the DiracDeltaNet [6]–[8]. ShiftNet and DiracDeltaNet use the shift operation, a hardware-friendly operation, that replaces the depthwise separable convolution, which introduces poor arithmetic intensity [9].

Latency-optimized approaches which report results on low-end low-cost devices are the Angel-Eye [10], the SqueezeJet-2 [11], and the latency-optimized version of fpgaConvNet [12].

In the taxonomy of the toolflows used for mapping CNN models on FPGAs presented in [1], FPGA accelerators are classified into two categories; the streaming architectures and the single computation engine architectures. Angel-Eye and

[1]https://github.com/pmousoul/cnn_grinder

SqueezeJet-2 fall in the single computation engine architecture category as opposed to fpgaConvNet which falls in the streaming architecture category. Single computation engine architectures execute the CNN layers in a time-shared manner and they don't require any FPGA re-configuration for accelerating different CNNs. On the other hand, streaming architectures consist of optimized hardware blocks which form a pipeline. This pipeline is used for the acceleration of a specific CNN; supporting a new CNN requires developing a new pipeline and performing an FPGA reconfiguration.

Many of the aforementioned works utilize a CNN compression method, such as quantization. A quantization strategy aims at reducing the CNN activation and parameter bitwidth without reducing the CNN accuracy. Dynamic fixed-point refers to fixed-point quantization which has different fractional part representation in different CNN layers. Angel-Eye utilizes a custom dynamic fixed-point quantization scheme which efficiently supports both 16 and 8-bit numbers. On the other hand, fpgaConvNet uses a 16-bit Q8.8 fixed-point arithmetic format. SqueezeJet-2 [11], [13] uses the dynamic fixed point format of Ristretto [14] allowing the design of both 8-bit and 16-bit accelerator implementations.

III. A Framework to Map CNNs to SoC FPGAs

This section aims to provide an introduction to the design of SqueezeJet-3 by presenting it in conjunction with the CNN-Grinder framework [13] which is used for mapping CNNs to SoC FPGAs. Unlike other CNN-to-FPGA frameworks, which hide the conversion of a CNN to HLS C/C++ behind automation, CNN-Grinder exposes this procedure to the user.

CNN-Grinder provides to the user Matlab code templates to convert the Caffe CNN description [15] to an algorithmic description. By using the Matcaffe interface of Caffe, the user can verify in Matlab the correct operation of the algorithmic CNN description. To efficiently map the CNN to an SoC FPGA, CNN input and parameters must be quantized. For this purpose, the Ristretto Caffe extension tool is used [14]. The user must now optimize the CNN in Matlab using code templates that support quantization. Using Matcaffe, the quantized algorithmic CNN implementation can be verified for correct operation against Ristretto. The next step is to transform the quantized algorithmic CNN description in order to be efficiently executed by the SqueezeJet-3 accelerator. This transformation includes input and parameter data dimension reordering, CNN layer reordering, and CNN layer operation reshaping. After verifying this last algorithmic implementation against Ristretto for correct operation, Matlab is used for the generation of binary files which hold the quantized input image, the quantized network parameters, and the inter-layer activation results. These files will be used by the HLS C/C++ implementation of the CNN. CNN-Grinder provides C/C++ and HLS C/C++ templates to ease the mapping of CNNs to SoC FPGAs.

The subsections that follow present the quantized parameter generation using Ristretto and the CNN transformations for accelerating the SqueezeNet v1.1 CNN using the SqueezeJet-3 accelerator.

A. CNN Quantization

SqueezeJet-3 is a CNN accelerator that can be configured to use operands with various accuracies. Since it is described in HLS C/C++, changing the operand accuracy is made as easy as modifying a couple of lines in a C/C++ header file. To be able to fit in low-end low-cost SoC FPGAs and at the same time provide high CNN inference accuracies, SqueezeJet-3 is configured to use 8-bit operands for the CNN activation and the CNN parameter data. These operands follow the dynamic fixed-point (DFP) format as it is defined by the Ristretto tool.

Table I shows the DFP parameters used for the quantization of the SqueezeNet v1.1 at 8 bits. These parameters are generated by Ristretto and subsequently are modified by the user to take into account the fact that the input activations to a concatenate layer must have the same DFP parameters. Another way to overcome this issue is to add layers for rescaling the inputs of a concatenate layer. The numbers in Table I show the per-layer fractional part of the input activation (**IN**), the output activation (**OUT**), and the layer parameters (**PAR**). A negative number indicates that the integer part of the value is larger than the bitwidth[2]. For example, in column **C1** the "-3" (**OUT**) quantization parameter indicates that the output activation of the first convolutional layer (**C1**) has no fractional part and that its value length requires 11 bits, 3 bits more than the bitwidth of the selected representation. This does not mean that the **C1** output activation requires 11 bits for its representation; what it means is that 8 bits are sufficient for representation and 11 bits are required to get the real values. Benchmarking SqueezeNet v1.1 in Ristretto using the quantization parameters of Table I results in a top-1 accuracy of 56.94%, which is less than 1.5% accuracy drop compared to the benchmarked floating-point top-1 accuracy (58.39%). The fine-tuning functionality of Ristretto could be used to decrease the accuracy drop below 1%; using fine tuning we achieve a top-1 accuracy of 57.42%. During fine-tuning the quantized CNN is trained to achieve increased accuracy.

Using operands in dynamic fixed-point format, the convolutional layer of a CNN can be described by:

$$A_o(y_o, x_o, c_o) = 2^{fo-(fi+fp)} \cdot MACC + 2^{fo-fp} \cdot B(c_o) \quad (1)$$

where $MACC$ is

$$MACC = \sum_{k_h=0}^{K_h-1} \sum_{k_w=0}^{K_w-1} \sum_{c_i=0}^{C_i-1} \{ A_i((y_o \cdot S + k_h), (x_o \cdot S + k_w), c_i) \cdot W(c_o, k_h, k_w, c_i) \} \quad (2)$$

and A_o, A_i are the output and the input activation respectively, W, B are the weight and bias parameters respectively, and fi, fo, and fp, are the input, output, and weight-bias quantization

[2]http://lepsucd.com/ristretto-cnn-approximation/ristretto-approximation-schemes/

TABLE I
ACTIVATION AND WEIGHT/BIAS PER-LAYER QUANTIZATION PARAMETERS FOR SQUEEZENET V1.1

LAYER	C1	F2/S1	F2/E1	F2/E3	F3/S1	F3/E1	F3/E3	F4/S1	F4/E1	F4/E3	F5/S1	F5/E1	F5/E3
IN	0	-3	-4	-4	-3	-3	-3	-3	-4	-4	-4	-4	-4
OUT	-3	-4	-3	-3	-3	-3	-3	-4	-4	-4	-4	-3	-3
PAR	7	6	7	7	7	7	7	6	7	7	7	7	7
LAYER	**F6/S1**	**F6/E1**	**F6/E3**	**F7/S1**	**F7/E1**	**F7/E3**	**F8/S1**	**F8/E1**	**F8/E3**	**F9/S1**	**F9/E1**	**F9/E3**	**C10**
IN	-3	-4	-4	-3	-4	-4	-3	-3	-3	-3	-3	-3	-2
OUT	-4	-3	-3	-4	-3	-3	-3	-3	-3	-3	-2	-2	-1
PAR	7	8	7	7	8	8	7	7	7	8	7	8	8

parameters respectively. y, x, c, represent the vertical, the horizontal, and the channel dimensions of the activations, S is the stride, and k_h, k_w are the vertical and horizontal dimensions of the filter.

SqueezeJet-3 can achieve the same accuracy as Ristretto by implementing the MAC operations of equation (2) using 8-bit registers for the input-output activations and the weights-bias values, 16-bit registers for holding the multiplication results, and 32-bit registers for holding the accumulation results. Assuming that no overflow occurs during the accumulation operation, the integer result of equation (2) retains its full accuracy.

A Ristretto-accurate result can be obtained by converting in equation (1) all the operands and operations to floating-point, calculating the result in floating-point, checking for over/under-flow at the 8-bit value boundaries, and truncating the result to 8 bits. By using special hardware modules which calculate the floating-point product with powers of 2, it is required only to convert the $MACC$ result and $B(c_o)$ value in floating-point, perform the products with the powers of 2 (using the special hardware modules), perform a floating-point addition, check for over/under-flow at the 8-bit value boundaries, and finally truncate the result to 8 bits. This hardware-friendly calculation method is used by the SqueezeJet-3 accelerator.

B. CNN Transformations

Some of the main innovations in the efficient mapping of the CNN computations to the SqueezeJet-3 accelerator are based on certain transformations. These transformations relate to the architecture and the operation of the accelerator.

(1) Activation and Parameter Data Dimension Reordering: Data are re-ordered from the Caffe format to the SqueezeJet-3 format. Activation data in the Caffe format are ordered as channel by height by width, $[C \times H \times W]$, and parameter data are ordered as output-channel by input-channel by filter-height by filter-width, $[C_o \times C_i \times K_h \times K_w]$. Activation data in the SqueezeJet-3 format are ordered as height by width by channel, $[H \times W \times C]$, and parameter data are ordered as output-channel by filter-height by filter-width by input-channel, $[C_o \times K_h \times K_w \times C_i]$. In all the aforementioned cases the rightmost dimension is the dimension that changes most frequently in the processing tasks. This dimension reordering is dictated by the 3D convolution operation; all the input channels corresponding to specific spatial locations of the input activation are required to calculate a single output channel of a specific spatial location of the output activation. The same holds in the case of filters; at least a single filter of size $[1 \times K_h \times K_w \times C_i]$ is required to calculate a single output channel of a specific spatial location of the output activation.

SqueezeJet-3 operates by calculating an output pixel[3] at a time by taking advantage of the available parallelism in the input and output channel dimensions. The input activation is streamed pixel by pixel in the accelerator and the weights-bias values are stored in the on-chip memory (OCM). If the OCM is not large enough, the weights-bias values are partitioned in the output channel dimension, and the whole input activation is streamed in the accelerator more than once, depending on the weight-bias partitioning.

(2) Layer Operation Reshaping: To overcome the first layer's reduced input channel issue [6] and to increase the accelerator's computational efficiency (which takes advantage of the parallelism in the input and output channel dimensions), the first layer's operation is reshaped. This reshaping converts a $n \times n, (n > 1)$ convolution to a 1×1 convolution with an increased input channel dimension. Since the input channel dimension is required by the accelerator to be a multiple of 16, zero-padding is often applied. Each processing element (PE) of the SqueezeJet-3 accelerator processes 16 values in the input channel dimension in parallel.

The layer reshaping operation is implemented in software and it is executed on the CPU of the SoC FPGA so as to take advantage of the very fast CPU cache, since it requires only memory operations. By performing this hardware/software partitioning, we reduce the latency required by the accelerator. Since each SqueezeJet-3 PE processes 16 values in the input channel dimension in parallel, it would require to zero-pad the input to become $[227 \times 227 \times 16]$ to process it without reshaping it. This would decrease the SqueezeJet-3 processing utilization since most of the values would be zeros (13 values from the 16 values in each input channel). Layer reshaping aims at improving the processing utilization of SqueezeJet-3.

(3) Layer Rearrangement: This transformation rearranges the max-pooling layers before the concatenate layers of the fire modules in the case of SqueeezeNet v1.1[4]. The reason is that the SqueezeJet-3 accelerator implements both the convolution and the max-pooling operations. Specifically, it consists of

[3]The word "pixel" is used to denote all the channels at a specific spatial location of the activation volume.

[4]https://dgschwend.github.io/netscope/#/preset/squeezenet_v11

two modules, the convolution and the max-pooling modules which are connected by a data streaming port and operate in a pipelined manner. By rearranging the max-pooling layers before the concatenate layers, data are not required to be written back to the main memory for the execution of both the convolution and the max-pooling operations; the data remains within the FPGA fabric and they are written back to the main memory after both operations have been completed.

IV. SqueezeJet-3

This section presents the architecture of the SqueezeJet-3 accelerator. SqueezeJet-3 is an extended version of a previous design [11]. The main modifications include improvements related to the transformations and the quantization methods listed in the previous section, as well as careful resource allocation of the BRAM and LUTRAM FPGA resources in order to provide better support for large CNNs, such as the VGG16 [16]. Additionally, the 2×2, with stride 2, max-pooling operation which is used by VGG16, is now supported by the SqueezeJet-3 design.

SqueezeJet-3 is a single computation engine architecture; it consists of a single computational block which is controlled by software and can accelerate CNN layers in a time-shared manner. The same design, which fits in low-end low-cost devices, such as the XC7Z020 and the ZU3EG, can be used to to accelerate CNNs ranging from SqueezeNet v1.1 to VGG16. SqueezeJet-3 provides acceleration for the following CNN layer types: convolutions with filter size 1×1 (padding 0), filter size 3×3 (padding 1 or 0, stride 1 or 2), ReLU non-linearities, 3×3 max-pooling with stride 2, and 2×2 max-pooling with stride 2. The accelerator's operation can be easily adapted to support additional layers since it is described in HLS C.

Other features of the accelerator include: (1) use of a double ping-pong buffering technique to overlap computation (MAC operations), input data communication (reading of new input activation pixel data), and output data communication (writing-back result pixels to the system's main memory), (2) DFP arithmetic support, (3) optional ReLU layer execution, and (4) high configurability; it is designed in a way that eases the configuration of (a) the number of OCMs and PEs used, and (b) the representation accuracy (in bits) of the data.

SqueezeJet-3 is designed using the Xilinx Vivado HLS tool [17] and it is interfaced to the processing system (PS) of the SoC FPGA using the Xilinx SDSoC tool [18]. The block diagram of the SqueezeJet-3 accelerator is shown in Figure 1. To optimize the activation and parameter data transfers, all the data are packed into software structs in order to take advantage of the full AXI port word length.

During the initialization phase, the Input Activation Rows (IACT-ROW) and the Parameters (Weights & Biases) OCMs are filled with data. The IACT-ROW OCM is filled with two rows of the activation data, in the case of a 3×3 filter size, and with no data in the case of a 1×1 filter size. The idea is to have enough data in the Input Activation Lines OCM and require only filter size number (e.g. 3, for a 3×3 filter) of input activation pixels to fill a convolution sliding Window OCM.

During the execution phase, where the CONV output activation volume is calculated pixel by pixel, additional input activation pixels are brought in the IACT-ROW OCM allowing to fill the Window OCM with the required data. To calculate the values of one output activation pixel and write them to one of the Output Pixel (OUT-PIXEL) OCMs, the Window OCM values are broadcasted to $16 \cdot 16 = 256$ registers. The Weights & Biases OCM, which is partitioned to 16 independent smaller OCMs, is providing to the PEs access to 256 parameter values. 16 PEs are executing concurrently 16 MACs in each clock cycle until all the data in the Parameters OCMs have been read, and one output activation pixel has been generated. Using double ping-pong buffering, during the calculation of one output activation pixel, the IACT-ROW OCM is filled with data, the Window OCM not in use is filled with data, and the output activation values calculated in the previous processing iteration are streamed from the OUT-PIXEL OCM which is not used at this time-period to the MAX-POOL part of the accelerator.

An AXI-Stream FIFO is transferring the CONV output activation pixels to the MAX-POOL part of the accelerator. The operation of the MAX-POOL part comprises of: (a) a channel-wise MAX operation between consecutive CONV output activation number of rows (this number is defined by the max-pooling filter size), and (b) a channel-wise MAX operation between the consecutive pixels of the row resulted in (a) - (again this number is defined by the max-pooling filter size). Two OCM pairs are defined for the aforementioned operation; the Current- and Result-Row OCMs are used in step (a) and the Current- and Result-Pixel OCMs are used in step (b).

V. Experiments

To test the performance and the accuracy of the SqueezeJet-3 accelerator, we benchmark it against DietChai, a miniature version of Xilinx's CHaiDNN v2 accelerator framework [19] designed for smaller Zynq/Zynq-UltraScale+ devices. Specifically, we build both SqueezeJet-3 and DietChai at 100MHz, we deploy them to the Ultra96 board (ZU3EG device), and we evaluate them onboard in terms of the CNN end-to-end inference latency, the CNN accuracy, and the power consumption. We use the SqueezeNet v1.1 CNN for making the aforementioned tests. Although CHaiDNN v2 advertises Ristretto support, Ristretto is not fully supported since no negative quantization parameters are allowed when using the dynamic fixed-point approximation scheme. To have a fair comparison in terms of accuracy, we benchmark on the Ultra96 board both DietChai and SqueezeJet-3 using the 50k ImageNet (ILSVRC2012) validation image set; in this benchmark, SqueezeJet-3 uses the Ristretto DFP approximation scheme and DietChai uses the quantization method developed by Xilinx.

To further test the performance of our accelerator, we use it to accelerate the convolutional layers of the VGG16 on both the ZC702 (XC7Z020 device) and the Ultra96 (ZU3EG

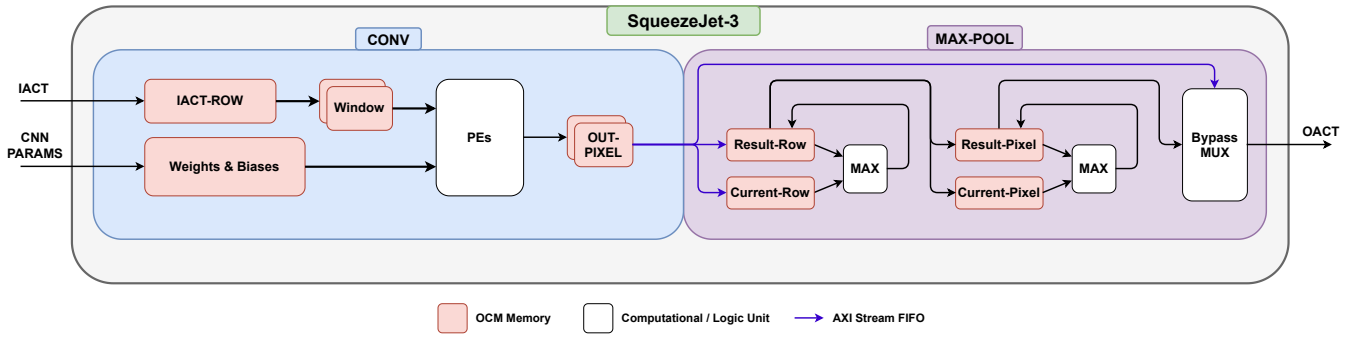

Fig. 1. Block diagram of the SqueezeJet-3 accelerator. Scalar arguments are omitted. The arrows show the direction of the data. Thicker arrow lines indicate block connections which can transfer more data than others.

TABLE II
LATENCY, RESOURCE UTILIZATION, AND ACCURACY RESULTS FOR SQUEEZENET V1.1

	DietCHai [19]	Our Work
FPGA device	ZU3EG	ZU3EG
Freq. (MHz)	100	100
Precision	8-bit	8-bit
Latency per frame (ms)	93	64
LUT	51948 (73.62%)	38468 (54.52%)
FF	69158 (49.01%)	35236 (24.97%)
BRAM	199.50 (92.36%)	124 (57.41%)
DSP	322 (89.44%)	270 (75.00%)
POWER (W)	4	2.6
ACCURACY (top-1: 58.39)	53.41	55.84

device) boards. We aim to compare SqueezeJet-3 against the Angle-Eye and the latency-optimized fpgaConvNet accelerators. We use the XC7Z020 device to run a 100MHz SqueezeJet-3 accelerator implementation and the ZU3EG device to run a 200MHz SqueezeJet-3 accelerator implementation.

VI. RESULTS & DISCUSSION

Table II shows the comparison between SqueezeJet-3 and DietChai when they are used for the acceleration of the SqueezeNet v1.1 CNN. The results show that SqueezeJet-3 is 31% faster in terms of latency. Even though DietChai is operating with a batch size equal to 2 and it requires 93ms for classifying 2 images, it still requires 93ms for providing a result for a single image. Additionally, SqueezeJet-3 requires less resources than DietChai for implementing this task; 19% less LUTs, 24% less FFs, 35% less BRAMs, and 14% less DSPs. Although with this amount of resources DietChai supports more layer types than SqueezeJet-3, it is unable to accelerate VGG16 on the Ultra96 board due to a limited buffer depth. As far as accuracy is concerned, Ristretto is a winner against Xilinx's quantization method by being 2.43% more accurate when both methods are benchmarked on the 50k ImageNet (ILSVRC2012) [20] validation set performed on the Ultra96v1 board[5]. Finally, the power consumption of SqueezeJet-3 is 40% lower than that of DietCHai.

Table III shows the comparison between SqueezeJet-3, Angel-Eye and fpgaConvNet, when they are used for the acceleration of the VGG16 CNN. As demonstrated by those results, even though SqueezeJet-3 does not support optimized convolution for 3×3 filters, its performance is very close to Angel-Eye and fpgaConvNet in terms of latency for the VGG16 CNN. Specifically, when operating at 214MHz, Angel-Eye is 20% faster than SqueezeJet-3 which operates at 200MHz. At 125MHz operational frequency, fpgaConvNet is very close to our 100MHz designs; fpgaConvNet is only 3% faster if we use linear interpolation to calculate the latency of our accelerator running at 125MHz on the XC7Z020 device; this is a valid claim since our design is processing-bound. The latency results can be explained by the fact that Angel-Eye and fpgaConvNet are optimized for the computation of a 3×3 filter size which is the same filter size used by the convolution layers of VGG16. On the other hand, SqueezeJet-3 is a 1×1 filter accelerator which emulates larger filters as 1×1 filters. Since 1×1 filter sizes are used very often in modern CNN architectures and especially in CNNs utilized in edge applications, this was a very careful design choice. Overall, we could report that our results are in line with the state of the art for both small and large CNNs. In Table III, the accuracy results of our work were obtained by benchmarking the DFP version of VGG16 using the Ristretto-based quantization utilized in SqueezeJet-3.

VII. CONCLUSIONS & FUTURE WORK

In this paper we present the architecture of a novel CNN accelerator called SqueezeJet-3. Our accelerator, which supports the dynamic fixed-point arithmetic format, uses 8 bits for representing both the activations and the CNN parameters, and it is optimized for accelerating CNNs which use both 3×3 and 1×1 filter sizes. SqueezeJet-3 fits into small low-end low-cost FPGA SoC devices, such as Xilinx's XC7Z020 and ZU3EG devices, it is suitable for edge CNN applications, it outperforms DietChai at the SqueezeNet v1.1 benchmark

[5]For a fair comparison, Ristretto fine-tuning was not performed in the SqueezeJet-3 accelerator case.

TABLE III
LATENCY, RESOURCE UTILIZATION, AND ACCURACY RESULTS FOR VGG16

	Angel-Eye [10]	fpgaConvNet [21]	Our Work	Our Work	Our Work
FPGA	XC7Z020	XC7Z020	XC7Z020	ZU3EG	ZU3EG
Freq. (MHz)	214	125	100	100	200
Precision	8-bit	16-bit	8-bit	8-bit	8-bit
Conv. Latency (ms)	364	633	874	831	447
LUT	29867 (56%)	?	44136 (82.96%)	38468 (54.52%)	38912 (55.15%)
FF	35489 (33%)	?	34011 (31.97%)	35236 (24.97%)	37555 (26.61%)
BRAM	85.5 (61%)	?	109.50 (78.21%)	124 (57.41%)	124 (57.41%)
DSP	190 (86.4%)	90.00%	220 (100.00%)	270 (75.00%)	334 (92.78%)
POWER (W)	3.5	?	2.6	2.6	3.1
ACCURACY (top-1: 68.32)	65.58	?	65.78	65.78	65.78

while its performance is very close to that of Angel-Eye and fpgaConvNet accelerators when executing the VGG16 benchmark; both Angel-Eye and fpgaConvNet are optimized explicitly for VGG16-like CNNs, where SqueezeJet-3 efficiently supports other CNN architectures as well. Future work will focus on enhancing the SqueezeJet-3 architecture so as to efficiently support the newer CNN approaches utilized in edge systems.

ACKNOWLEDGMENTS

This research is financed by Greece and the European Union (European Social Fund - ESF) through the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant" (Project Number: 2198).

REFERENCES

[1] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, p. 56, 2018.

[2] F. Iandola and K. Keutzer, "Small neural nets are beautiful: enabling embedded systems with small deep-neural-network architectures," in *Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion*. ACM, 2017, p. 1.

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

[4] D. Gschwend, "Zynqnet: An fpga-accelerated embedded convolutional neural network," *Master Thesis ETH-Zurich: Swiss Federal Institute of Technology Zurich*, 2016.

[5] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[6] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer, "Squeezenext: Hardware-aware neural network design," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1638–1647.

[7] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, "Shift: A zero flop, zero parameter alternative to spatial convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9127–9135.

[8] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek *et al.*, "Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019, pp. 23–32.

[9] K. Kwon, A. Amid, A. Gholami, B. Wu, K. Asanovic, and K. Keutzer, "Co-design of deep neural nets and neural net accelerators for embedded vision applications," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[10] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

[11] P. G. Mousouliotis and L. P. Petrou, "Software-Defined FPGA Accelerator Design for Mobile Deep Learning Applications," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2019, pp. 68–77.

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

[13] P. G. Mousouliotis and L. P. Petrou, "CNN-Grinder: From Algorithmic to High-Level Synthesis descriptions of CNNs for Low-end-low-cost FPGA SoCs," *To be appeared in: Microprocessors and Microsystems*, 2020. [Online]. Available: http://doi.org/10.1016/j.micpro.2020.102990

[14] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks." IEEE, 2018.

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] Xilinx, "Vivado Design Suite User Guide - High-Level Synthesis," https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug902-vivado-high-level-synthesis.pdf, 2018, [Online; Accessed 14-September-2019].

[18] V. Kathail, J. Hwang, W. Sun, Y. Chobe, T. Shui, and J. Carrillo, "SDSoC: A Higher-level Programming Environment for Zynq SoC and Ultrascale+ MPSoC," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 4–4.

[19] Xilinx, "A HLS-based Deep Neural Network Accelerator library for Xilinx Ultrascale+ MPSoC devices." Xilinx, 2018. [Online]. Available: https://github.com/Xilinx/CHaiDNN

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[21] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: mapping regular and irregular convolutional neural networks on FPGAs," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 326–342, 2018.