Lowering Dynamic Power of a Stream-based CNN Hardware Accelerator

Duvindu Piyasena*, Rukshan Wickramasinghe†, Debdeep Paul‡, Siew-Kei Lam§ and Meiqing Wu¶
*,†,§, ¶Nanyang Technological University, Singapore
‡Indian Institute of Technology, Patna
{*gpiyasena, §assklam,¶meiqingwu}@ntu.edu.sg, †wmr.rukshan@gmail.com, ‡debdeep.ee15@iitp.ac.in

***Abstract*—Custom hardware accelerators of Convolutional Neural Networks (CNN) provide a promising solution to meet real-time constraints for a wide range of applications on low-cost embedded devices. In this work, we aim to lower the dynamic power of a stream-based CNN hardware accelerator by reducing the computational redundancies in the CNN layers. In particular, we investigate the redundancies due to the downsampling effect of max pooling layers which are prevalent in state-of-the-art CNNs, and propose an approximation method to reduce the overall computations. The experimental results show that the proposed method leads to lower dynamic power without sacrificing accuracy.**

***Index Terms*—CNN, FPGA acclerator, low-power designs**

Convolutional Neural Networks (CNNs) are finding its way into a wide range of embedded applications such as image classification [1], object detection [2], semantic segmentation [3], etc. The use of CNNs in embedded devices must meet multiple conflicting design constraints such as performance, power consumption and cost. Custom hardware implementations (such as implementations on Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs)) offer good opportunities to meet these constraints compared to their software counterparts (CPUs or GPUs).

It has been shown that a major bottleneck in existing CNN architectures lies in the memory accesses to transfer feature maps and weights between the external memory and processing engine [4]. A stream based CNN hardware accelerator [5] offers the advantage of high throughput processing and avoids the need for using external memory for storing the intermediate feature maps. The intermediate feature maps are streamed from one layer to another in a pipeline manner using internal row buffers.

The elimination of external memory accesses in a fully stream based CNN architecture leads to faster execution time, but incur significant computational resources as all the CNN layers are executed concurrently. This leads to high dynamic power dissipation due to the intensive computations. In particular, the convolution layers account for the majority of computations in a typical CNN (over 90%) [6], and hence it has the highest impact on the power consumption. This work aims to reduce the dynamic power consumption of a stream based CNN architecture by removing the computational redundancies in the convolutional layers due to the downsampling effect of the max pooling operations. Max pooling is found in most state-of-the-art CNNs and is used to downsample feature maps by selecting the maximum activation in a neighborhood and discarding the rest. This incurs high computational redundancies in the convolution operations that compute the activations which are eventually discarded in max pooling.

We propose to remove these redundancies by predicting the feature map candidates in the neighbourhood that will result in maximum activation prior to performing convolution. The prediction is achieved with simple approximations. The actual convolution operation is only performed on these candidates. This eliminates the computational redundancies at the convolution layers, by avoiding convolutions that result in activations which would eventually be discarded by max pooling. We introduce a methodology that is able to derive suitable approximations for different CNN models by analyzing the distribution of the weights. Using the proposed method, we show that significant number of redundant convolution operations can be avoided in various CNN models. We also integrated the proposed approximation in a stream based CNN architecture that is generated by Haddoc2 [5]. FPGA power analysis results show that the proposed scheme leads to 33% reduction in dynamic power while preserving the original accuracy.

I. Related Work

A. Streaming Hardware Architectures

A common technique to implement CNNs on FPGAs is to design a uniform accelerator that is time-shared to perform sequential layer-by-layer processing [7], [8]. However, this approach produces large intermediate data between layers that requires off-chip memory storage. This leads to high off-chip memory accesses and the runtime of CNNs being constrained by the off-chip memory access bandwidth. The latency and throughput degradations will violate the real time requirements of CNNs . High off-chip memory accesses also consume high power [9], leading to lower energy efficiency. Additionally, using a uniform accelerator with fixed configuration to process all layers lead to resource under-utilization as different layers have varying inputs, outputs and feature map dimensions.

Streaming CNN accelerators built upon dataflow processing paradigm, have been presented in [10]–[18] to address the above-mentioned issues. The underlying premise of stream-based accelerators is to maximize on-chip memory usage and eliminate/reduce the off-chip memory accesses in order to alleviate performance and energy bottlenecks of the time-

shared implementations. Some of the key features associated with most stream-based accelerators are:

- Inter-layer parallelism, with pipelined processing of multiple/all layers on-chip. Each layer starts processing once sufficient data is available. Hence, intermediate data storage requirements can be significantly reduced.
- Use of on-chip buffers to stream data between layer.
- Multiple heterogeneous accelerator units, each optimized for a particular layer configuration for resource utilization efficiency.

However, the implementation of stream based accelerators is challenging due to on-chip resource and memory constraints. These challenges have been addressed by previous work via various techniques.

The work in [16], [17] maps multiple heterogeneous convolution layers, to parallel heterogeneous units. In [10], a given CNN is partitioned to sub dataflow graphs, with seperate bitstreams functioning via reconfiguration. These approaches are optimized for high throughput than latency.

In latency driven approaches, streaming accelerators are developed using multiple layer fusion [14], [15], and changing tile granularity [13], with single/multiple FPGA approaches. In [11], quantized CNNs (QNN), are implemented as fully unrolled streamed accelerator. Recently FPGA clusters are used to deploy fully unrolled designs hardened for layer configurations [12]. In addition to the above-mentioned FPGA based approaches, the work in [18] demonstrated an ASIC multi-die realization of stream-based acceleration.

The above-mentioned work have demonstrated that streaming CNN accelerator offers the potential for achieving high throughput and runtime performance compared to the uniform accelerator approach. One key driving factor that will enable stream-based processing of CNNs will be the continuous growth of on-chip memory and compute power in FPGAs. For instance, Xilinx Ultrascale+, the latest generation of Xilinx FPGAs are equipped with a new on-chip memory, Ultraram, resulting in 6x increase in on-chip storage capacity (enabling up to 500Mb on-chip storage) than the previous generation [19]. Our work aims to reduce the power consumption of streaming CNN accelerators by reducing the convolution operations, which accounts for majority of power dissipation.

B. CNN Hardware Optimization

Optimizing compute efficiency of CNN accelerators is an actively researched area. Existing work can be broadly classified as follows.

1) Reducing Precision: Bitwidth quantization is an actively explored area in CNNs to remove overheads associated with full precision arithmetic. It has been shown that CNNs are resilient to fixed point forms with linear quantization [20], logarthmic quantization [21] and even extreme forms such as binarization [22]. Alternatively, encoding schemes too have been proposed [23]. These approaches allow efficient computations/storage in CNN hardware accelerators compared to floating point implementations.

2) Network Pruning: Network pruning is another popular technique to reduce the redundant operations in CNNs. The objective is to compress the network, by sparsifying network connectivity [24], [25] or by structured pruning to eliminate structures such as filters/channels/layers, altogether [26].

3) Approximate Computing: Approximate Computing is an emerging area to approximate computations via low-precision forms to gain performance benefits [27], [28]. The work in [27] replaces less critical neuron computations via approximated forms, while [28] extends this to reduce memory accesses. However using approximations to replace computations leads to accumulation of errors causing accuracy degradations in larger networks.

Our work employs lightweight approximations to predict redundant convolution computations so that they can be eliminated. This enabled us to achieve significant computational savings while maintaining original accuracy without the need to retrain the network. Similar approach has been taken in [29], [30] to eliminate convolution redundancies caused by max-pooling. However, these work have been demonstrated only on smaller datasets like MNIST and CIFAR10. We have evaluated our method on much larger datasets like Imagenet, which consists of 1000 object classes for classification.

II. Proposed Methodology

The proposed method relies on simple approximations to determine the convolution candidates that will result in maximum activation in a pooling neighborhood. These approximations are derived from the weight distribution of the CNN. In this section, we describe the proposed methodology for determining the approximation scheme for a given trained CNN model.

A. CNN Layers

CNNs typically consists of a series of convolutional (CONV) and Pooling layers, followed by several Fully Connected (FC) layers at the final stages. CONV layers extract features hierarchically in the form of feature maps by convolving the input feature maps with two-dimensional pre-trained filters. Activation functions (eg : ReLU, TanH) are used to introduce non-linearity to CONV feature maps. The POOL layers are used for representative feature selection and down-sampling feature map representations to reduce computational complexity. There are two types of POOL layers: average pooling (AVG) and max-pooling (MAX).

The Max Pool operation in particular performs a neighborhood operation on CONV feature maps, where the maximum activation of each neighborhood is propagated while the rest of activations is discarded, making the computation effort incurred for the discarded activations redundant. The objective of this work is to eliminate this redundancy to gain power savings in hardware.

The following parameters are used to describe the configurations of CNN layers:

- N_i : Number of input feature maps(IFM) in the layer
- N_o : Number of output feature maps(OFM) in the layer
- N_r : Number of rows in the input feature map
- N_c : Number of cols in the input feature map

- K_c : Dimension of Convolution kernel
- S_c : Stride of Convolution Kernel
- K_p : Dimension of Pool Kernel
- S_p : Stride of Pool Kernel

B. Redundancy Definition

The objective of the proposed work is eliminate the compute redundancy occuring as a result of the max-pool operation explained above. The following equations characterize the redundancy of CONV activations and hence the FLOPS (multiplications and addition operations), caused due to Max Pool, in a given CONV-MAX layer arrangement.

Max Pool IFM dimension (N_{conv}) = $N_r \times N_c$
Max Pool OFM dimension (N_{pool}) = ($N_r \times N_c$) / S_p^2
FLOPS per each IFM activation = $2 \times N_r \times N_c \times K_c^2 \times N_i$
FLOPS discarded/Layer =
$2 \times N_r \times N_c \times K_c^2 \times (1 - 1/S_p^2) \times N_i \times N_o$
Redundancy (%) = (N_{conv}-N_{pool})/N_{conv} = (1 - 1/S_p^2)

For this, we have assumed the Input Feature map dimensions of Max Pool and CONV layers are equal (i.e. S_c = 1). To evaluate this redundancy, we used several popular CNN models, Lenet [31], Cifar10-Quick [32], Cifar10-Full [31], Cifar10-NiN [33], Alexnet-BN [34] and VGG16 [6]. The redundancy in CONV-MAX layers reaches upto 75%, for layers with S_p = 2, which is a common configuration seen across models. Table I, indicates the evaluated redundancy, in which row 6, indicates the redundancy of each CONV-MAX layer as a percentage of total computations.

C. Approximation Method

The proposed method aims to eliminate computational redundancies occuring due to Max Pool effect explained above, to obtain power and energy savings in hardware. We propose a lightweight approximation method, that predicts the required CONV activations prior to actual computation, hence avoiding the redundant computations. The approximation scheme predicts the CONV neighborhoods which produce Max Pool candidates, and the CONV operations are performed only for the predicted neighborhoods. This is illustrated in Fig. 1. In the rest of the paper, we refer to this approximation scheme for CONV operations as ***'ApproxConv'*** and the neighborhoods predicted by the approximations as ***'Max Pool Candidate Neighborhoods (MPCNs)'***.

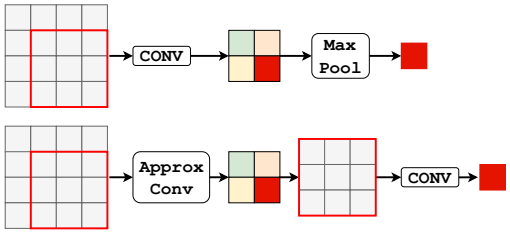
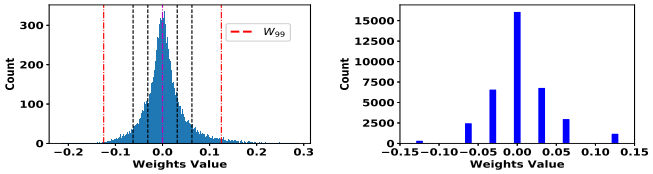


Fig. 1: Conventional method vs Proposed method



(a) Weight distribution (b) ApproxConv weights

Fig. 2: ApproxConv weights quantization in VGG16 2nd layer

To achieve the objective of power gains via this method, the ApproxConv operation must be a low complexity implementation, such that the power benefits obtained by this removing redundant CONV operations are not outweighed by the ApproxConv overhead. Hence this work propose ApproxConv to approximate CONV operations using original weights quantized to power-of-two values. This allows ApproxConv to use low cost Bit-shifters to approximate multiplication operation in hardware. Additionally this work investigates the most compact power-of-2 representations for ApproxConv weights via a static analysis across each layer in a given model, to further simplify the ApproxConv units, and maximise power gains. This static analysis of mapping original weights to ApproxConv weights is explained in the next subsection.

D. Convolution Weights Approximation

In a typical CONV layer the weights are distributed between -1 and +1, as shown in Fig. 2a. Hence the weights of ApproxConv can be represented by power-of-2 quantization levels, $\pm 1/2^n$ where n $\in$ Z$\geq$ 0. To arrive at the most compact number of levels per each layer, we perform a static analysis on the trained model, using the validation dataset. An iterative search is performed to identify the optimal level count with accuracy loss under 0.5%. The steps in this analysis are,

1) The number of power-of-2 levels ($\boldsymbol{N_L}$) of all layers are initially set to 4.
2) In each layer, the original weights are clipped at 99th percentile to remove outliers. The closest power-of-two value to this clipping point is chosen as the Maximum quantization level. In a given layer, we define this point as '$\boldsymbol{W_{99}}$', with the exponent as '$\boldsymbol{m}$'.
3) Hence, given $\boldsymbol{N_L}$ and $\boldsymbol{m}$, quantization levels for ApproxConv weights are set to 0, $\pm 1/2^m$, $\pm 1/2^{m+1}$, ..., $\pm 1/2^{m+N_L-1}$.
4) Each weight is mapped to the nearest power-of-2 level defined in the above step
5) The modified design is tested in Caffe for accuracy using validation image set
6) Steps 2-5 are repeated for all the combinations of $\boldsymbol{N_L}$ across layers.

After repeating steps 2-5, the optimal configuration with the accuracy drop under 0.5% is chosen as the optimal weights mapping for ApproxConv. Then the exponents of levels are packed as ApproxConv weights. In a given layer the maximum bitwidth of ApproxConv weights is log2($\boldsymbol{N_L}$ + 1).

TABLE I: Computation Redundancy caused by Max Pool in popular CNN models

	Lenet-5		Cifar10-quick [?]	Cifar10-full [?]	NiN	VGG16					BN-AlexNet		
	Conv-Max1	Conv-Max2	Conv-Max1	Conv-Max1	Conv-Max1	Conv-Max1	Conv-Max2	Conv-Max3	Conv-Max4	Conv-Max5	Conv-Max1	Conv-Max2	Conv-Max3
CONV activations/Layer	11,520	3,200	32,768	32,768	98,304	3,211,264	1,605,632	802,816	401,408	100,352	290,400	186,624	43,264
FLOPS/Layer	576,000	3,200,000	4,915,200	4,915,200	31,457,280	3,699,376,128	3,699,376,128	3,699,376,128	3,699,376,128	924,844,032	210,830,400	895,795,200	299,040,768
FLOPs Discarded/layer	432,000	2,400,000	3,686,400	3,686,400	23,592,960	2,774,532,096	2,774,532,096	2,774,532,096	2,774,532,096	693,633,024	160,022,016	688,128,000	235,339,776
Discarded FLOPS (% by Layer))	**11.44%**	**63.56%**	**75.00%**	**75.00%**	**75.00%**	**17.65%**	**17.65%**	**17.65%**	**17.65%**	**4.41%**	**11.38%**	**48.95%**	**16.74%**

Fig. 2, shows the weights distribution of 2nd CONV layer of VGG16, where dotted lines represent the weights of ApproxConv weights and red dotted lines represent W_{99} at the iteration when N_L = 3. In this scenario the ApproxConv weights are mapped to $\{0, \pm 0.03125, \pm 0.0625, \pm 0.125\}$, as indicated in Fig. 2b.

III. Hardware Architecture

This section contains the implementation details of the hardware architecture of the above discussed approximation method. The implementation is based on the Haddoc2 [5], an open source tool capable of generating FPGA based CNN accelerators automatically, from a Caffe model.

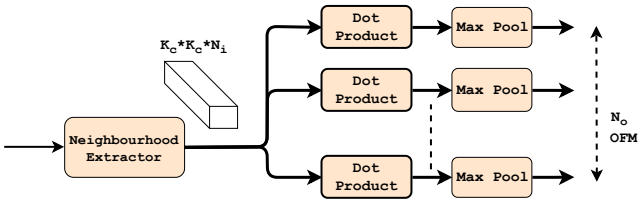
A. Baseline Design

The baseline hardware generated from Haddoc2 is a dataflow graph with streamed processing computations carried out in 8-bit integer precision. The weights are hardcoded, to allow Synthesis optimizations to multipliers.

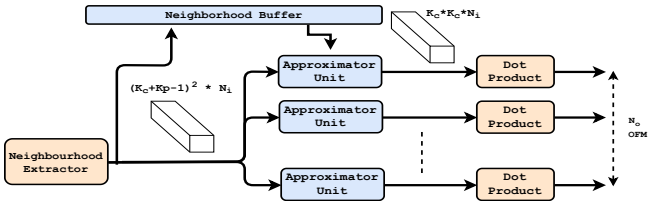
A single convolution layer in Haddoc2 is indicated in Fig. 3a.The important modules in the Haddoc2 are,

- ***Neighborhood Extractor*** : The neighborhood extractor buffers the incoming pixels in the row buffers and outputs convolution neighborhoods at every clock cycle. Row buffers are N_r in length and are implemented as registers.
- ***Dot Product Unit*** : Each Dot Product unit corresponds to a CONV operation of an OFM. It is a loop unrolled hardware unit with K_c*K_c*N_i number of parallel multipliers followed by an adder tree containing same number of adders. Hence it takes an input tensor of dimension K_c*K_c*N_i and outputs a convoluted pixel. A convolution layer contains N_o parallel dot product units.
- ***Max Pool*** :
  The Max Pooling unit is implemented as a combination of vertical and horizontal max pool units which performs max pooling across vertical and horizontal dimensions respectively. The vertical pool unit buffers inputs from K_p-1 rows and outputs the vertical maximums. These vertical maximums are fed into horizontal pooling unit, which buffers the previous K_p-1 inputs and outputs the horizontal maximums.
- ***Activation*** : ReLU was added to Haddoc2, which performs a simple sign based multiplexing operation.

The hardware generated by Haddoc2 for the original network configuration in Caffe, was used as the baseline design for our comparisons.



(a) Baseline Hardware Architecture (single layer)



(b) Modified Hardware Architecture (single layer)

Fig. 3: Hardware Architectures of baseline and modified designs (single layer)

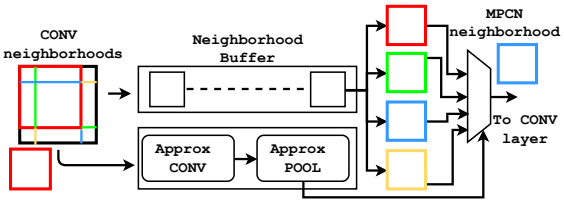


Fig. 4: Approximator Unit

B. Modified design

The modified design, contains additional units namely the Approximator Units and neighborhood buffer as indicated in Fig. 3b.

The Approximator units performs the ApproxConv operation and selects the MPCNs for which Dot Product units perform CONV operations. Thus the Approximator units precedes the dot product units in hardware. The subcomponents of the Approximation unit are as follows,

- ***ApproxConv*** : The ApproxConv unit is similar in structure to Dot Product unit with several differences to make it lightweight. The multiplications are replaced by bit-shift operations due to proposed power-of-two quantized weights. Since the weights are hardcoded, the bit-shifters are further optimized to bit-concatenations during synthesis. Additionally using compact power-of-two weights results in low-cost adders compared to the original Dot Product units.
- ***Approx Pool*** : The Approx Pool performs the max pooling on the ApproxConv outputs. The output selections are used to drive the neighborhood multiplexer at the output stage to produce MPCNs, to be produced for actual CONV operation.

The internals of the Approximator unit is shown in Fig. 4.

The neighborhood buffer, delays CONV neighborhood inputs to dot product units, to synchronize with the outputs of Approx Pool selections mentioned above. The neighborhood buffer can buffer neighborhoods of N_i number of IFMs. For each IFM, neighborhood buffer outputs the CONV neighborhoods of a given Max-Pool neighborhood parallely, which are fed to the neighborhood multiplexer. The neighborhood multiplexer selects the MPCN, from the input neighborhoods based on the Approx Pool outputs.

The Dot Product units are clock gated in the design, such that the reduced activity leads to reduction of power dissipation. Since the operation of the Approximator units are pipelined and overlapped with other processing the performance is not significantly impacted. Thus the proposed design leads to significant energy gains following the power gains observed.

IV. Experimental Results

This section contains experimental results of the accuracy evaluation and the hardware synthesis of the proposed method.

A. Accuracy Evaluation

The accuracy of the proposed method was evaluated on image classification datasets for CNNs mentioned in Section II, using Caffe [35]. Top-1/Top-5 accuracies on the validation image set were used as metrics for accuracy evaluation.

We perform an iterative search by varying the number of power-of-two levels for ApproxConv weights according to the steps outlined in section II D. The mapping which gives the least number of levels, with accuracy loss under 0.5% is chosen as the optimal weights mapping for ApproxConv. Table II, summarizes the final weights mapping results and accuracies across the tested models. The 4th column, contain the level counts of ApproxConv weights.

Apart from comparing accuracy with original models, we also compared our method with the method proposed in [29]. The method in [29], uses sign of weights to approximate CONV operation and is hence identified as *SignConnect*. For comparison, we applied *SignConnect*, to the same networks, and the accuracies are reported in Table II. The results indicate that *SignConnect*, gives mixed results, with significant losses in accuracy for AlexNet-BN, Cifar10-Full. Comparatively, our method arrives at low-complex complex approximations, for all the reported models, with accuracy drop under 0.5%. Hence the proposed scheme, is a better method to approximate CONV operations for Max Pool predictions.

B. Hardware Evaluation

The baseline and the modified designs were implemented using VHDL by extending Haddoc2. Fully Connected Layers were excluded from the implementations, as the focus of this work is on CONV layers, which accounts to the majority of computations. The designs were synthesized at 100MHz clock frequency, using Vivado 2018.2 targeting Virtex Ultra-Scale+ VCU1525 Acceleration Development Board (xcvu9p-fsgd2104-2L-e).

The dynamic power consumption was measured from the synthesized designs using switching activity generated from post-synthesis simulations, run on Modelsim 10.6C. The power consumption was measured for MNIST samples covering all digits and average power figures are reported.

TABLE III: Hardware Evaluation Results

		Baseline	Modified	Change (%)
Dynamic Power (W)		1.919	1.289	**-32.83%**
Resource	**LUT**	431752	814558	88.66%
	FF	156178	317096	103.03%
Latency (ns)		7980	8010	0.38%
Energy/Image (J)		1.53E-05	1.03E-05	**-32.58%**

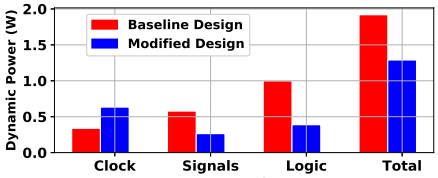


Fig. 5: Dynamic Power Consumption Breakdown

Table III, reports the power and utilization figures of the baseline and modified designs. The power analysis results in Table III shows that there is dynamic power improvement of 32.83% in the modified design with approximation. Additionally, due to the overlapped processing of the approximation operations, the latency increase is marginal. Hence the energy/image is improved by 32.58%, in the modified design.

However, this benefit comes at the expense of 88.66% increase in LUT and 103.03% increase in FF resource consumption than the baseline design. The increase in LUT resources is mainly due to the neighborhood multiplexing units explained in section III.B. The FF count increases as a result of the neighborhood buffers.

The breakdown of dynamic power consumption is shown in Fig. 5. Although the clock power has increased in the modified design as a result of increase in FF, the logic and signal powers have reduced by considerable margins, owing to the reduction of redundant CONV operations, through approximations.

V. Conclusions

This work presented a method to lower dynamic power of a streaming based CNN hardware accelerator. This is achieved by exploiting run-time redundancies in the convolution layers that is due to the Max Pooling operations. The experimental results shown that significant dynamic power reduction of 33% can be achieved with the proposed method, while not sacrificing on the accuracy. Potential future work includes evaluating proposed method on larger CNN implementations and investigating methods to lower the increase in resource usage in the proposed hardware design.

TABLE II: Accuracy Evaluation

Network	Baseline Accuracy (Top-1/5)	Sign Connect	Proposed Method		
			Nl (By Layer)	Power-of-2 Levels (By Layer)	Accuracy (Top-1/5)
VGG16	68.15/88.14	67.99/87.78	3 2 2 2	0.0625, 0.125, 0.250 0.031250, 0.0625 0.015625, 0.03125 0.015625, 0.03125	68.02/88.09
AlexNet-BN	56.57/ 79.92	21.13/40.60	4 2 3	0.03125, 0.0625, 0.125, 0.25 0.03125, 0.0625 0.015625, 0.03125, 0.0625	56.11/79.37
Cifar10-Quick	72.19/97.70	70.88/97.81	1	0.125	71.87/97.69
Cifar10-Full	81.66/99.12	74.53/98.53	2	0.125, 0.25	81.42/99.07
Cifar10-NiN	89.57/99.62	89.43/99.62	2	0.25, 0.5	89.49/99.62
Lenet	99.01/99.99	99.05/99.99	2 1	0.5, 0.25 0.125	99.05/100

VI. Acknowledgement

This work was supported in part by the National Research Foundation Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) Programme with the Technical University of Munich at TUMCREATE.

References

[1] 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*, vol. 115, no. 3, pp. 211–252, Dec 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, June 2017.

[3] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, April 2017.

[4] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct 2013, pp. 13–19.

[5] K. Abdelouahab, M. Pelcat, J. Srot, C. Bourrasset, and F. Berry, "Tactics to directly map cnn graphs on embedded fpgas," *IEEE Embedded Systems Letters*, vol. 9, no. 4, pp. 113–116, Dec 2017.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[7] 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

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," 02 2015, pp. 161–170.

[9] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 10–14.

[10] S. I. Venieris and C. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 40–47.

[11] C. Baskin, N. Liss, E. Zheltonozhskii, A. M. Bronstein, and A. Mendelson, "Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018, pp. 162–169.

[12] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. El Husseini, T. Juhasz, K. Kagi, R. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A. Rapsang, S. Reinhardt, B. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P. Yi Xiao, D. Zhang, R. Zhao, and D. Burger, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, Mar 2018.

[13] X. Wei, Y. Liang, X. Li, C. H. Yu, P. Zhang, and J. Cong, "Tgpa: Tile-grained pipeline architecture for low latency cnn inference," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2018, pp. 1–8.

[14] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[15] Q. Xiao, Y. Liang, L. Lu, and S. Y. and, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[16] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 535–547.

[17] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang, "A high performance fpga-based accelerator for large-scale convolutional neural networks," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–9.

[18] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014, pp. 609–622.

[19] Ultraram: Breakthrough embedded memory integration on ultrascale+ devices. [Online]. Available: https://www.xilinx.com/support/documentation/white$_$papers/wp477-ultraram.pdf

[20] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *CoRR*, vol. abs/1604.03168, 2016. [Online]. Available: http://arxiv.org/abs/1604.03168

[21] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 5900–5904.

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[24] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage.pdf

[25] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1135–1143. [Online]. Available: http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf

[26] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *CoRR*, vol. abs/1608.03665, 2016. [Online]. Available: http://arxiv.org/abs/1608.03665

[27] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Aug 2014, pp. 27–32.

[28] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 701–706.

[29] T. Ujiie, M. Hiromoto, and T. Sato, "Approximated prediction strategy for reducing power consumption of convolutional neural network processor," in *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2016, pp. 870–876.

[30] M. Ahmadi, S. Vakili, J. M. P. Langlois, and W. Gross, "Power reduction in cnn pooling layers with a preliminary partial computation strategy," in *2018 16th IEEE International New Circuits and Systems Conference (NEWCAS)*, June 2018, pp. 125–129.

[31] Lenet, caffe example. [Online]. Available: https://github.com/BVLC/caffe/tree/master/examples/mnist

[32] Cifar10 example networks. [Online]. Available: https://github.com/BVLC/caffe/tree/master/examples/cifar10

[33] Network in network, cifar10. [Online]. Available: https://gist.github.com/mavenlin/e56253735ef32c3c296d

[34] Caffe, "Alexnet-bn, caffe," Feb. 2018. [Online]. Available: https://github.com/HolmesShuan/AlexNet-BN-Caffemodel-on-ImageNet

[35] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.