# Reducing Dynamic Power in Streaming CNN Hardware Accelerators by Exploiting Computational Redundancies

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*—Convolutional neural networks (CNNs) have achieved tremendous successes in various application domains such as computer vision. However, current implementations are characterized by large memory requirements and accesses, which pose an impediment towards their deployment on low cost embedded devices with fast runtime requirements. Recently, FPGA based streaming CNN hardware accelerators have been reported for alleviating these memory bottlenecks. However, these implementations suffer from large number of convolution operations which incur high power consumption. In this paper, we investigate methods to exploit the redundancies in the activation layers in order to reduce the dynamic power. We propose a computationally efficient approximation method to reduce the overall convolution operations with marginal accuracy loss. Experimental results of our FPGA implementation based on image classification datasets show that the proposed method leads to considerable power savings.**

***Index Terms*—Convolutional neural networks, FPGA, Streaming hardware accelerators, dynamic power reduction, approximate computing**

## I. Introduction

Deep learning using convolutional neural networks(CNNs) is finding its way into a wide range of Internet-of-Thing(IoT) applications. These applications are often characterized by large-scale data streaming that requires real-time processing at the IOT device itself while meeting power constraints [1].

Recently, there is a growing interest in realizing custom hardware accelerators on Field Programmable Gate Arrays (FPGAs) to alleviate some of the above-mentioned challenges [2]. It has been shown that the major performance bottleneck in existing CNN accelerators lies in its dependence on off-chip memories to store intermediate computations and weights [3]. Streaming-based hardware accelerators [4]–[16] offer opportunities for achieving higher performance by eliminating external memory accesses. However, they also 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 this paper, we show that a considerable proportion of convolution activations are discarded as a result of activation functions, leading to high computational redundancies. Our work aims to reduce dynamic power consumption of streaming CNN hardware accelerators, by removing these redundant computations.

To achieve this, we present a hardware friendly convolution approximation scheme which predicts required computations prior to actual computations. We show that the proposed method leads to higher computational savings for wider CNN models/datasets without sacrificing on accuracy compared to existing approaches [17], [18]. The experimental results based on streaming FPGA based accelerator indicates, that by using the proposed method, power savings of over 12% and energy savings over 11% can be achieved.

## II. Related Work

### A. Streaming Hardware Architectures

FPGA based CNN accelerators are commonly implemented in the form of a time-shared uniform accelerator which processes layers sequentially [19], [20]. The large intermediate data produced between layers are stored off-chip. However, this approach suffers from performance and energy degradations [21], due to high off-chip memory accesses and resource underutilization due to heterogeneous CNN layers [22].

Alternatively, streaming CNN accelerators, built upon dataflow processing paradigm, with reduced/eliminated off-chip storage and maximized on-chip buffering have been proposed in [4], [6]–[8], [11]–[16] to address these issues.

Most stream based CNN accelerators are characterized by,

- Inter-layer parallel processing.
- Use of on-chip buffers to stream data between layers.
- Heterogeneous accelerator units, for better utilization.

However, the implementation of stream based accelerators is a challenging task due to on-chip resource and memory constraints. In existing work, stream based CNN accelerators have been realized by various techniques: using bitstream reconfiguration [4], fusing multiple layers [12], [13], changing tile granularity [11], quantized CNNs (QNN) [7] and more recently using FPGA clusters [6], [8]. In addition to the FPGA based approaches, the work in [16] demonstrated an ASIC multi-die realization of stream-based acceleration.

One key driving factor that will enable stream-based processing of CNNs will be the continuous growth of on-chip memory and compute resources in FPGAs [23]. 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 CNNs is an actively researched area. Existing work can be 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 encoding based quantization schemes [24], fixed point forms with linear quantization [25], logarthmic quantization [26] and even extreme forms such as binarization [27].

*2) Network Pruning:* This is another popular technique to reduce the redundant operations in CNNs. The objective is to compress the network, by sparsifying network connectivity [28], [29] or by structured pruning to eliminate structures such as filters/channels/layers, altogether [30].

*3) Approximate Computing:* This is an emerging area to approximate computations via low-precision forms to gain performance benefits [31], [32]. However using approximations to replace computations lead 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 [17], [18] 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 larger datasets like Imagenet [33]. Additionally, we show that compared to these methods, the proposed method leads to potentially higher computational savings.

## III. Background and Motivation

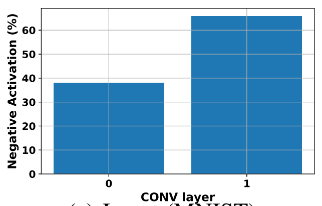
### A. CNN Layers

CNNs typically consist of a series of convolution (CONV), activation (ACT) and pooling (POOL) 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 2D pre-trained filters. Activation functions are used to introduce non-linearity to CONV feature maps. Most popular CNN models use ReLU activation function denoted by (1). The POOL layers are used for representative feature selection and downsampling feature map representations to reduce computational complexity.

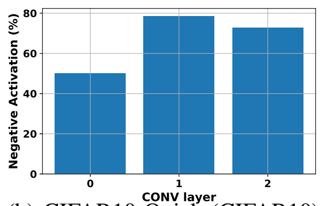
$$ReLU : f(x) = max(0, x) \tag{1}$$

### B. Redundancy Analysis

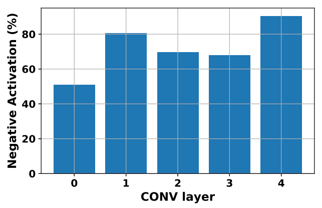
According to (1), ReLU sets all the negative input CONV activations to zero. Hence, the computational effort incurred for CONV operations for negative activations become redundant. To evaluate the extent of this redundancy in well-known CNN models, we used Caffe [34] to measure the percentage of negative activations in several pre-trained CNN models: Lenet [35] (MNIST [36]), Cifar10-Quick [37] (CIFAR10 [38]), AlexNet [39] (Imagenet [33]) and VGG16 [40] (Imagenet [33]). Fig. 1 shows the percentage of negative CONV activations in each layer for the above-mentioned networks. It can



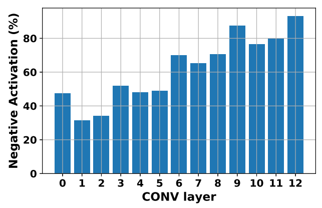
(a) Lenet (MNIST)



(b) CIFAR10-Quick (CIFAR10)



(c) AlexNet (Imagenet)



(d) VGG16 (Imagenet)

Fig. 1. Percentage of negative activations by layer

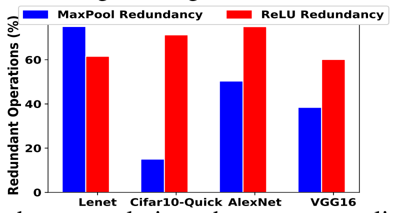


Fig. 2. Redundant convolutions due to max-pooling and ReLU

be observed that the redundancy in certain layers range from 30%-90%, with the latter layers exhibiting significantly higher redundancy. Thus, eliminating CONV operations that will result in negative activations provides tremendous opportunity to achieve computational and power savings.

We also compared the potential savings of our method, with the previous approach that eliminates redundant CONV operations due to max-pooling [17], [18]. Fig. 2, shows the percentage of redundant computations across all networks mentioned above. It can be observed that the proposed method results in higher potential computational savings in all the networks except for Lenet. This is due to the fact that apart from Lenet, in larger networks max-pooling form only a fraction of total operations compared to activations.

## IV. Proposed Methodology

### A. Overview

The proposed method aims to eliminate the computational redundancies arising from ReLU activation by predicting the positive/negative CONV activations using a low cost approximation scheme. The regular CONV operations are performed only on the predictions yielding positive activations. In particular, the proposed CNN layer consists of the following stages:

1) ***ApproxConv*** : Lightweight approximation is performed to obtain approximated CONV activations.
2) ***ReLUpred*** : Sign of approximated CONV activations is checked.
3) ***CONV*** : Actual CONV operations are performed for inputs/feature maps associated with positive predictions and zeros are assigned to activations associated with negative predictions.
4) ***ReLU*** : Finally, ReLU is applied to the CONV activations to eliminate false-positives.

The proposed method aims to eliminate redundant computations in order to achieve power and energy savings in hardware. However, this also incurs additional hardware units that are required for the prediction. As such, the ApproxConv operation must lend itself well towards low-complexity implementation such that the power consumed by ApproxConv does not outweigh the power gained from removing the redundant CONV operations. In the next subsection, we describe our methodology to determine low-cost ApproxConv units.

## *B. Model-specific Convolution Approximation*

To achieve a low-cost ApproxConv unit, we approximate the convolution operations using original weights that are quantized to power-of-two values represented by $\pm 1/2^n$ where n $\in$ Z$\geq$ 0. The mapping of original CONV weights to ApproxConv weights is done by performing a static analysis on the trained model of the original network using the validation data set. The steps in the proposed methodology are:

1) Initialize number of power-of-two quantization levels ($\boldsymbol{N_L}$) of all layers to 8.
2) The weights are saturated at the 99th percentile to remove the adverse effects of outliers on the mapping. The closest power-of-two value to this saturation point is chosen as the maximum quantization level. We define this point as '$\boldsymbol{W_{99}}$', and the exponent as '$\boldsymbol{m}$'.
3) At the start of each iteration, the 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 assigned to the nearest quantized level and the modified model is tested in Caffe for accuracy using the validation image set.
5) Steps 3-4 are repeated by decrementing $\boldsymbol{N_L}$ at each iteration till it reaches 1.

Finally, the weights mapping with the lowest $\boldsymbol{N_L}$ whose accuracy loss is under 1% is chosen and the corresponding exponents are packed as the ApproxConv weights, which requires a maximum bitwidth of $log2(2 * \boldsymbol{N_L} + 1)$.

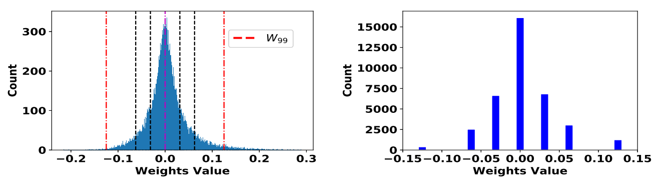
Fig. 3a shows the weights distribution in the second CONV layer in VGG16, where the dotted lines represent the ApproxConv quantization levels while the red dotted line represent '$\boldsymbol{W_{99}}$' at the iteration when $\boldsymbol{N_L}$=3. At this instance, the ApproxConv quantized weights are mapped to {0, $\pm$0.03125, $\pm$0.0625, $\pm$0.125}, as indicated in Fig. 3b.

## V. Proposed Streaming CNN Accelerator

The proposed CNN architecture is based on stream-based processing where all the layers are computed in parallel without external memory accesses. The outputs from one layer is streamed to the next via on-chip buffers. The layers are executed in parallel, where each layer starts processing once it receives sufficient pixels for a valid CONV window.

## *A. Baseline Design*

This subsection describes the hardware architecture of the baseline design that will be used to evaluate the proposed method. The architecture uses 8-bit integer precision for activations and weights. The default quantization is based on the offline approach used in Nvidia TensorRT [41]. The trained weights are hardcoded to allow synthesis optimizations of



(a) Weight distribution (b) ApproxConv weights

Fig. 3. ApproxConv weights quantization in VGG16 2nd layer

multipliers. As the focus of this work is on CONV layers, FC layers were excluded from the design.

The parameters used to describe the configurations of CNN layers are, $\boldsymbol{N_i}$, $\boldsymbol{N_o}$, which represent number of input and output feature maps of a layer respectively. $\boldsymbol{K_c}$ and $\boldsymbol{K_p}$, represent dimensions of the Convolution and Pool kernels respectively, while $\boldsymbol{S_p}$ represents pool stride. The height and width of input feature maps are represented by $\boldsymbol{H}$ and $\boldsymbol{W}$, respectively.

*1)* ***Line Buffer:*** This unit caches the incoming pixels and outputs convolution neighborhoods at every clock cycle. Each unit contains ($\boldsymbol{K_c}$-*1*) $\times$ ***Ni*** row buffers for buffering convolution neighborhoods of ***Ni*** feature maps. We assume that the pixels are read in a raster scan manner and hence, the row buffers are $\boldsymbol{W}$ in length and they are implemented as registers.

*2)* ***CONV Unit:*** A loop-unrolled design consisting of parallel multipliers followed by an adder tree. Each CONV unit takes an input of dimension $\boldsymbol{K_c} \times \boldsymbol{K_c} \times \boldsymbol{N_i}$ and outputs a convolved pixel. Each unit contains $\boldsymbol{K_c} \times \boldsymbol{K_c} \times \boldsymbol{N_i}$ number of parallel multipliers and adders. A convolution layer contains $\boldsymbol{N_o}$ parallel CONV units.

*3)* ***Max Pool:*** The Max-pooling unit is decomposed into two units performing max-pooling across vertical and horizontal dimensions respectively. The vertical pool unit buffers inputs from $\boldsymbol{K_p}$-1 rows and outputs vertical maximums. These are fed into horizontal pooling unit, which buffers the previous $\boldsymbol{K_p}$-1 inputs and outputs horizontal maximums.

*4)* ***ReLU:*** ReLU performs a multiplexing operation on the pixel stream based on the sign bit of the input pixels.

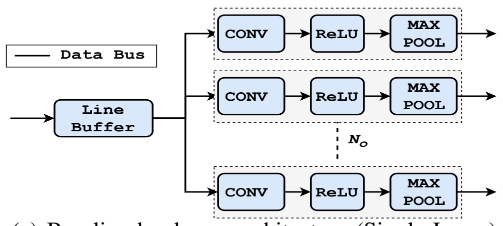
A single layer of the baseline architecture with the configuration CONV-ReLU-MAX is shown in Fig. 4a.

## *B. Proposed Design*

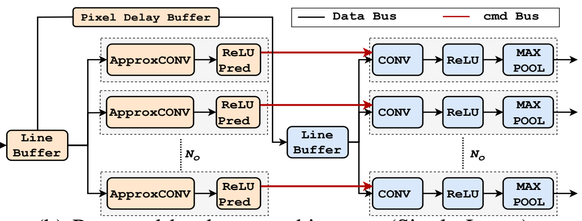
This proposed hardware architecture is shown in Fig. 4b (only a single layer is shown). The following describes the additional hardware units that are integrated in the proposed achitecture to eliminate redundant CONV operations.

*1)* ***ApproxConv:*** The ApproxConv unit is similar in structure to the CONV unit with several key differences to make it lightweight. The multipliers are replaced by bit-shift operations due to the proposed power-of-two quantized weights. As the weights are hardcoded, the synthesis tool optimizes the bit-shifters to bit-concatenation operations. The use of the proposed power-of-two quantized weights also result in the instantiation of low-cost adders.

*2)* ***ReLUPred:*** An extension of the normal ReLU unit which provides no-operation commands (NO-OP) to the subsequent CONV operations (required for convolution of feature maps associated with predicted positive activations). The No-OP signals are used to clock gate CONV circuitry.



(a) Baseline hardware architecture (Single Layer)



(b) Proposed hardware architecture (Single Layer)

Fig. 4. Hardware architectures of a single layer in baseline and proposed designs

TABLE I. Accuracy comparisons

| Network | Baseline | SignConnect [17] | | Proposed | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | SignConnect-1 | SignConnect-2 | Prop-1 | | | Prop-2 | | |
| | Accuracy (Top-1/Top-5) | Accuracy (Top-1/Top-5) | Accuracy (Top-1/Top-5) | Level count | Weight Bitwidth | Accuracy (Top-1/Top-5) | Level count | Weight Bitwidth | Accuracy (Top-1/Top-5) |
| **VGG16** | 68.15/88.14 | 39.62/64.34 | 40.11/65.49 | 3 | 3 | 67.94/87.65 | 3 | 3 | 67.67/87.67 |
| **AlexNet** | 56.57/79.92 | 27.08/50.75 | 32.98/58.01 | 5 | 4 | 56.3/79.5 | 3 | 3 | 55.77/79.35 |
| **CIFAR10-Quick** | 72.19/97.69 | 68.18/97.03 | 68.95/96.97 | 3 | 3 | 71.74/97.53 | 2 | 3 | 71.88/97.75 |
| **Lenet** | 99.08/100 | 98.97/100 | 99.02/100 | 2 | 3 | 99.099/100 | 1 | 2 | 99/100 |

*3) **Pixel Delay Buffer**:* A synchronization buffer is used to delay the inputs for the original CONV layer to process after the ApproxConv and ReLUPred have completed operations.

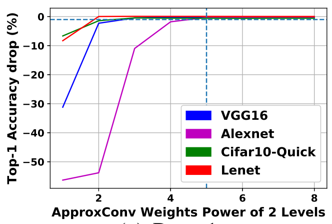
## VI. Experimental Results

### *A. Accuracy Evaluation*

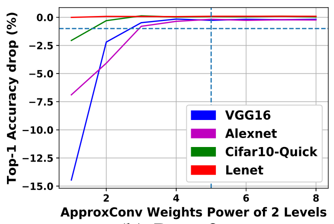
The accuracy of the proposed method was evaluated based on Top-1/5 accuracies using CNN models and datasets mentioned in Section II.

Fig. 5 shows the change in Top-1 accuracy for varying number of power-of-two levels for ApproxConv weights according to steps in Section III. As expected, error becomes high for lower number of levels. As it was empirically found out that using approximations for first layer notably increases the error at lower number of levels, experiments were performed applying approximations to all layers (*Prop-1*) and all layers except the first layer (*Prop-2*). The results of *Prop-1* and *Prop-2* shown in Fig. 5a and Fig. 5b respectively, indicates that when approximations are omitted from first layer error at lower number of levels reduces.

Table I compares the accuracy of our methods *Prop-1* and *Prop-2*, with other methods. Columns 5, 6 and columns 8, 9 shows the optimized number of levels and the corresponding bitwidth of the weights for *Prop-1* and *Prop-2* respectively. In addition to the *Baseline* (no approximations are applied), we also compare our accuracy with [17]. The method in [17] uses the sign of the weights to perform the approximations and we denote the method as *SignConnect*. Similar to our approach, we evaluated the case where the method in [17] is applied to all layers (*SignConnect-1*), and to all layers except the first (*SignConnect-2*). It can be observed that although *SignConnect* performs well for Lenet and Cifar10-Quick networks, error is too high for Alexnet and VGG16. In comparison, our method results in marginal accuracy drop for all the networks.



(a) Prop-1



(b) Prop-2

Fig. 5. Top-1 accuracy change with power-of-two levels

### *B. Hardware Evaluation*

The baseline and proposed designs (*Prop-1* and *Prop-2*) for Lenet [35] were implemented using Verilog HDL. The optimal configurations shown in Table I were used for the proposed method. The designs were synthesized at 100Mhz, targeting Xilinx Virtex Ultrascale+ xcvu9p device, using Vivado 2018.3. The dynamic power consumption was measured using switching activity generated from post-synthesis simulations, run on Modelsim 10.6C, for MNIST samples covering all digits and average power figures are reported.

The Table II summarizes the power, resource consumption and the latency for the designs. It can be observed that *Prop-1* and *Prop-2* achieve dynamic power gains of **10.79%** and **12.17%** respectively over *Baseline*. Since the performance degradation is minimal, the energy/image gains over *Baseline* for *Prop-1* and *Prop-2* are **10.03%** and **11.83%** respectively. However, this benefit comes at the expense of slight increase in LUTs due to additional ApproxConv units and an increase in register counts due to additional row buffers.

TABLE II. Hardware evaluation

| | | Baseline | Prop1 | | Prop2 | |
|---|---|---|---|---|---|---|
| | | | | Change(%) | | Change(%) |
| **Dynamic Power (W)** | **Total** | 2.2057 | 1.9565 | **10.79%** | 1.9263 | **12.17%** |
| | **Conv** | 1.2749 | 0.9357 | **18.91%** | 0.9509 | **19.00%** |
| | **ApproxConv** | 0 | 0.0943 | - | 0.0779 | - |
| | **Other** | 0.9308 | 0.9265 | -0.46% | 0.8975 | 2.76% |
| **Resource** | **LUT** | 627269 | 685650 | 9.31% | 680867 | 8.54% |
| | **FF** | 297106 | 394420 | 32.75% | 391079 | 31.63% |
| | **BRAM** | 28 | 31 | 10.71% | 30.5 | 8.92% |
| **Latency (ns)** | | 9130 | 9210 | 0.88% | 9165 | 0.38% |
| **Energy/Image(J)** | | 2.00E-4 | 1.80E-04 | **10.03%** | 1.77E-04 | **11.83%** |

## VII. Conclusions

This work presents a method to lower dynamic power of streaming CNN FPGA accelerators by exploiting runtime redundancies in the convolution layers due to ReLU activation operations. The proposed method leads to run time computational and power savings with minimal accuracy and performance loss. Future work may consider evaluations on larger networks where savings are expected to be high.

## VIII. Acknowledgement

This research project is partially funded by the National Research Foundation Singapore under its Campus for Research Excellence and Technological Enterprise(CREATE) programme with the Technical University of Munich at TUM-CREATE.

## References

[1] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2923–2960, Fourthquarter 2018.

[2] Y. LeCun, "1.1 deep learning hardware: Past, present, and future," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, Feb 2019, pp. 12–19.

[3] 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.

[4] 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.

[5] W. Jiang, E. H. Sha, Q. Zhuge, L. Yang, X. Chen, and J. Hu, "Heterogeneous fpga-based cost-optimal design for timing-constrained cnns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2542–2554, Nov 2018.

[6] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: ACM, 2016, pp. 326–331. [Online]. Available: http://doi.acm.org/10.1145/2934583.2934644

[7] 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, IPDPS Workshops 2018, Vancouver, BC, Canada, May 21-25, 2018*, 2018, pp. 162–169. [Online]. Available: https://doi.org/10.1109/IPDPSW.2018.00032

[8] 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.

[9] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 1–14.

[10] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Patel, and M. Herbordt, "A framework for acceleration of cnn training on deeply-pipelined fpga clusters with work and weight load balancing," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2018, pp. 394–3944.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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

[20] 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.

[21] 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.

[22] Y. Shen, M. Ferdman, and P. Milder, "Overcoming resource underutilization in spatial cnn accelerators," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[23] Ultraram: Breakthrough embedded memory integration on ultrascale+ devices. [Online]. Available: https://www.xilinx.com/support/documentation/whitepapers/wp477-ultraram.pdf

[24] 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

[25] 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

[26] 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.

[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, 2016. [Online]. Available: http://arxiv.org/abs/1603.05279

[28] 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

[29] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015. [Online]. Available: http://arxiv.org/abs/1506.02626

[30] 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

[31] 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.

[32] 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.

[33] 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

[34] 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.

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

[36] Y. LECUN, “The mnist database of handwritten digits,” *http://yann.lecun.com/exdb/mnist/*. [Online]. Available: https://ci.nii.ac.jp/naid/10027939599/en/

[37] Ciffar10 example networks. [Online]. Available: https://github.com/BVLC/caffe/tree/master/examples/cifar10

[38] A. Krizhevsky, “Learning multiple layers of features from tiny images,” *University of Toronto*, 05 2012.

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

[40] 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

[41] Nvidia tensorrt 8-bit quantization. [Online]. Available: http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf