A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks

(Invited Paper)

Vinayak Gokhale*, Jonghoon Jin*, Aysegul Dundar†, Berin Martini† and Eugenio Culurciello†
*Electrical and Computer Engineering, Purdue University
†Weldon School of Biomedical Engineering, Purdue University

***Abstract*—Deep networks are state-of-the-art models used for understanding the content of images, videos, audio and raw input data. Current computing systems are not able to run deep network models in real-time with low power consumption. In this paper we present nn-X: a scalable, low-power coprocessor for enabling real-time execution of deep neural networks. nn-X is implemented on programmable logic devices and comprises an array of configurable processing elements called collections. These collections perform the most common operations in deep networks: convolution, subsampling and non-linear functions. The nn-X system includes 4 high-speed direct memory access interfaces to DDR3 memory and two ARM Cortex-A9 processors. Each port is capable of a sustained throughput of 950 MB/s in full duplex. nn-X is able to achieve a peak performance of 227 G-ops/s, a measured performance in deep learning applications of up to 200 G-ops/s while consuming less than 4 watts of power. This translates to a performance per power improvement of 10 to 100 times that of conventional mobile and desktop processors.**

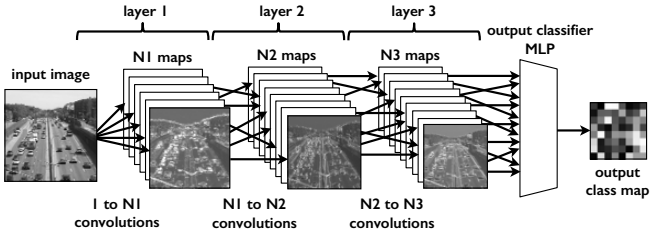

Figure 1: Architecture of a typical convolutional deep neural network for object recognition: a convolutional feature extractor followed by a classifier (like a multi-layer perceptron) for generic multi-class object recognition. Once trained, the network can parse arbitrarily large input images, generating a classification map as the output.

I. Introduction

The next grand challenge for mobile devices is to be able to understand the world in the same way we do. By understanding their environment, these devices will be able to provide a new ecosystem of abilities to increase user perception and connect user preferences to human knowledge. Perceptive mobile devices should be able to parse and understand relationships between objects in a scene. Perceptive algorithms can enhance speech processing and allow always-on hearing capabilities. Such devices will be able to understand the world around them and allow verbal interaction with users, just like two human beings.

Deep neural networks that achieve state-of-the-art perception in both vision and auditory systems have been presented in [11], [13]. Deep networks are models that make sense of raw input data and parse them into symbols. Many deep learning algorithms are loosely inspired models of the human visual system. These algorithms use convolution operations to model the receptive fields of real neurons [16].

A typical deep neural network comprises multiple convolution layers followed by a classification module, as portrayed in Figure 1. The convolution layers are interspersed with a pooling operation and a non-linearity. The inputs and outputs of each layer are sets of arrays called feature maps. Each feature map represents a particular feature extracted at all locations for that layer's input. Deep neural networks are currently considered the state of the art in speech and image processing, as exemplified by the recent use of these techniques by Google, Facebook, Yahoo and Baidu, to name a few [11], [13].

Deep neural network models are computationally very expensive, requiring up to billions of operations per second [11]. Typically, high performance processors like server CPUs and GPUs are needed to process large deep networks in real-time. This makes computation on mobile devices prohibitive, especially when running on battery power. Mobile processors like the ARM Cortex-A9 and Cortex A-15 have higher performance per unit power, but they are not able to scale to the raw computational performance needed to run deep networks in real-time.

In this paper we present *nn-X: neural network next*. nn-X is a low-power mobile coprocessor for accelerating deep neural networks and is optimized to process multiple streams of information. The coprocessor efficiently implements pipelined operators with large parallelism, thus delivering very high performance per unit power consumed. nn-X advances the state-of-the-art in multiple domains: in data-streaming architectures, in efficient processing of deep neural networks, in providing high performance in real applications and ultimately, in efficient use of system power using standard digital circuits and programmable logic devices.

The rest of this document is organized as follows: Section II reports related literature and results in this area, Section

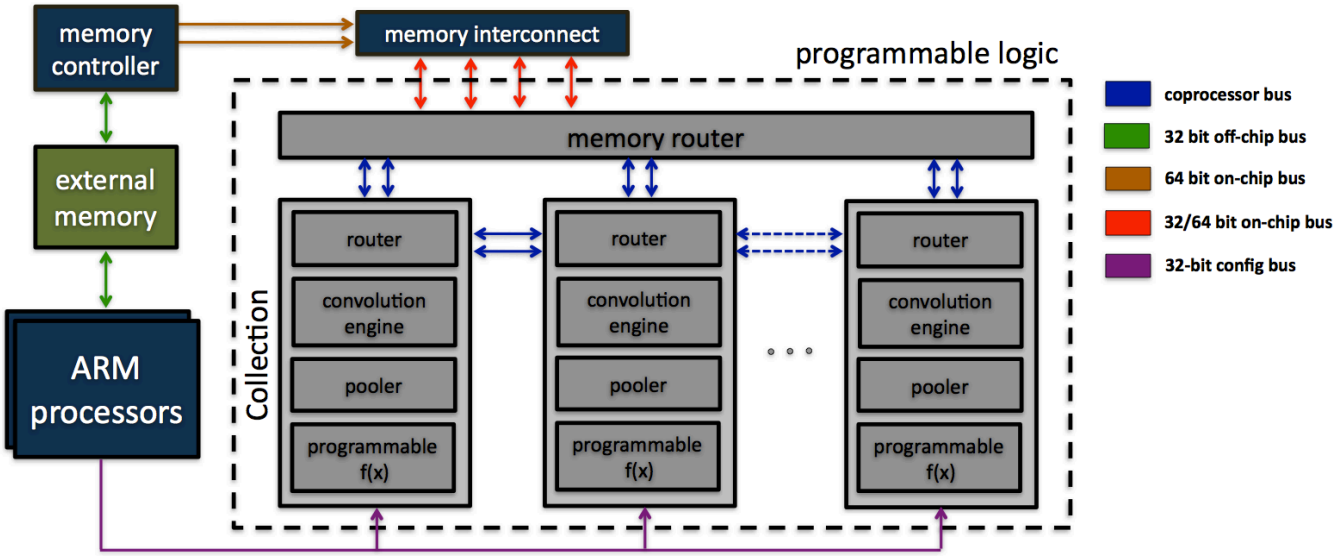

Figure 2: A block diagram of the nn-X system. nn-X is composed of a coprocessor, a host processor and external memory. The coprocessor has three main components: processing elements called collections, a system bus called the memory router and a configuration bus to control flow of data. Collections perform the most typical deep neural network operations: data routing, convolutions, pooling, non-linear programmable functions.

III describes the system architecture in detail, Section IV shows the results of running real applications on nn-X, Section V contains a discussion on why nn-X outperforms general purpose processor implementations and Section VI concludes this paper.

II. Related Work

Neural network accelerators for images, videos and other type of data systems have been implemented on FPGAs before. In [9], Graf, et.al. describe a processor for accelerating, training and running neural networks. This work is designed for generic machine learning algorithms. However, the results provide low performance for specific network models, like deep neural networks. In addition, it targeted desktop computers. Newer work by the same team on deep neural networks claims a theoretical performance of 23 G-ops/s but can achieve 6.74 G-ops/s in a real application [15]. A more recent work by Chakradhar demonstrates a dynamically reconfigurable coprocessor but their peak performance and performance per watt are significantly lower than the work presented here [2] .

A similar FPGA implementation has been previously documented by Farabet in [6] and [7]. This system uses ethernet to communicate to a host computer, making it inefficient. The architecture also does not scale as well as nn-X. Furthermore, nn-X is 2 to 15 times faster than the works mentioned above while consuming 8 watts for the entire platform and only 4 watts for the coprocessor, memory and host.

ASIC implementations have been described by Mesa-Camunas [1] and Chen [3]. [1] is a low power accelerator but is limited to processing 128×128 inputs. [3] describes a complete neural processor but their performance is measured only when processing small inputs. Memory access latency is not taken into account.

III. System Architecture

A block diagram of the nn-X system is shown in Figure 2. nn-X has three main components: a host processor, a coprocessor and external memory. The coprocessor comprises an array of processing elements called *collections*, a memory router and a configuration bus. The collections are a group of mathematical operators required to perform the most typical operations of deep neural networks.

A. The Host Processor

Two ARM Cortex-A9 CPUs function as the host processor for the nn-X implementation described here. The processor is responsible for parsing a deep network, compiling it into instructions for the coprocessor and processing operations that are not implemented in the programmable logic. The host also controls transfer of input and configuration data to the coprocessor.

B. nn-X Coprocessor

The nn-X coprocessor is implemented on programmable logic and interfaces with the host via the AXI bus. Input data in the form of 2D planes is streamed into the nn-X coprocessor, one data word per clock cycle. Data is organized

as an array, with data words streamed in one row at a time. These data words can be pixels in the case of images or videos. This section describes in detail the components of the nn-X coprocessor.

1) Collections: Each collection comprises of: one convolution engine, one pooling module and one non-linear operator. All operators use the Q8.8 number format, which has been tested to provide virtually identical results to neural networks implemented in 32-bit floating point [9], [3], [10]. As can be seen in Figure 2, each collection also has an internal router to direct input data to the desired operator or to neighboring collections. Each operator is pipelined which results in one output word produced every clock cycle, notwithstanding an initial setup time. The following paragraphs define the flow of data in a generic deep network of the type described in [11].

Convolution engine: Convolutions are the most typical operation in deep and convolutional neural networks. Convolution is inherently parallel and can be accelerated on data parallel architectures. The operation is described here:

$$y[m,n] = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} x[m+i, n+j] \cdot w[i,j] \qquad (1)$$

where $y[m,n]$ is one data word in the output plane, $x[m,n]$ is an input data word and $w[m,n]$ are the weights in the filter kernels.

When a convolution needs to be performed, the weights are first streamed in. These weights are cached for the duration of the convolution. The nn-X implementation described here supports a kernel size of up to 10×10.

The convolution engine is implemented as fully pipelined logic and uses memory to cache incoming data. This cache is needed for pipelined implementation of the convolution operation [8]. For a row width of W and a $k \times k$ convolution filter, the size of this cache is $W \times k \times 2$ bytes. After a delay that is equal to the depth of this cache, outputs are available every clock cycle. This requires the system to have the max allowable data width as a design parameter.

Output data can then be routed to other operators in the collection to perform cascaded pipelined sequences of operations. It can also be sent to a neighboring collection to be combined with the output of a convolution performed there.

The convolution engine can also perform pooling operations. The kernel can be used to smooth the pooling function (for example, Gaussian) or perform a simple averaging of pixels or data words (with a uniform kernel).

Non-linear operator: The non-linear operator computes a piecewise linear approximation of any arbitrary non-linear function. The non-linear operation is described in equation (2).

$$y(x) = a_m x + b_m \quad \text{for} \quad x \in [l_m, l_{m+1}) \qquad (2)$$

where l_m is the lower bound of the m-th segment and a_m and b_m are its slope and y-intercept.

The non-linear operator can be programmed to approximate the typical non-linear functions used in deep neural networks like sigmoids, hyperbolic tangents and the recently popular threshold operation, i.e. $\max(x, 0)$ [11].

The number of linear segments used is also a design parameter. This affects the precision of smooth, non-linear function approximations.

Pooling module: Pooling of data is necessary in deep neural networks to increase spatial invariancy and reduce the size of data after each layer. nn-X includes a special max-pooling module that calculates the maximum over a 2D region of $p \times p$ data words and outputs the maximum value as the result.

nn-X's max-pooling module requires one digital comparator and memory. The input data is streamed into the module one row at a time and the max operation is computed by first computing the max of each $\lfloor \frac{W}{p} \rfloor$ group of data words in the row, with W being the width of each row. As data words are streamed in, each group of p is compared to the previous max value stored in memory. This requires storing $\lfloor \frac{W}{p} \rfloor$ values into a local cache, as the first output cannot be computed until the first p data words of the p-th row are streamed in. After operating on p rows, the final output can be computed and output values start to stream out of the module.

The advantage of this implementation is that it requires a very small amount of memory to compute the maximum over a 2D region. In fact, the total memory required is equal to the maximum width of the input image.

2) Memory Router: The memory router interfaces the collections with external memory. Its purpose is to route independent data streams and feed data to the collections. The router is implemented as a crossbar switch, allowing nn-X access to multiple memory buffers at once and performing full-duplex data transactions. It interfaces with the Zynq's AXI memory interconnect, which allows for up to four DMA channels with an aggregate bandwidth up to 3.8 GB/s.

DMA transactions to and from memory are initiated by a custom Linux device driver. This driver enables nn-X to initiate up to four simultaneous bidirectional transactions at a given time. nn-X uses register polling to determine when a transaction is complete.

3) Configuration Bus: The collections and the memory router are configured via a 32-bit memory-mapped configuration bus. 32 registers are implemented to configure all parameters needed by nn-X to perform operations. The host processor reads the compiled configuration data from memory and writes it on the config bus. The configuration bus programs the memory router to multiplex inputs to multiple collections. It also programs all parameters of the convolution engine, the non-linear functions and the flow of data within and across collections. This bus is also controlled

by a custom Linux device driver. The time taken to transfer one packet is 16 clock cycles.

IV. Results

The nn-X implementation described in this paper was prototyped on the Xilinx ZC706 platform (refer to Table I).

This board contains two ARM Cortex-A9 cores, 1 GB of DDR3 memory and a large programmable logic array. nn-X on the ZC706 board features eight collections, each with one 10×10 convolution engine, one max-pooling module and one non-linear mapping module. We measured the power consumption of the entire board to be 8 W and 4 W for the Zynq SoC and DDR3 memory.

The ZC706 platform was chosen because performance increases linearly with the number of collections, and being able to fit 8 collections gave us a favorable balance of performance and performance per watt.

Platform	Xilinx ZC706
Chip	Xilinx Zynq XC7Z045 SoC
Processor	2 ARM Cortex-A9 @800 MHz
Programmable Logic	Kintex-7
Memory	1 GB DDR3 @533MHz
Memory bandwidth	3.8 GB/s full-duplex
Accelerator frequency	142 MHz
Number of Collections	8
Peak performance	227 G-ops/s
Power consumption	4 W (Zynq+mem), 8 W (board)

Table I: This table describes nn-X's hardware specifications

Torch7 was used as the main software tool in this work [5]. Torch7 is a module implemented in the Lua programming language. It is a machine learning tool optimized for nn-X, CPUs and GPUs. We developed demonstration applications for neural networks in Torch7 and used a Lua/Torch7 interpreter to translate the networks into configuration sequences for nn-X.

Figure 3: Single neural network layer with 4 input planes of 500×500, 18 outputs planes and 3.6 billion operations per frame. nn-X computed one frame in 6.2 ms and was 271 times faster than the embedded processors.

We measured the performance of nn-X and compared it to that of the Zynq's dual ARM cores in multiple applications. Figure 3 shows a still from the first application: a fully-connected neural network layer with 4 inputs and 18 outputs. The network used 10×10 convolution kernels with 4×18 random filters, a max-pooling operation of 4×4 and thresholding. This network required 3.6 billion operations per frame. In this application, nn-X computed one frame in 6.2 ms and achieved a speed-up of 271x with respect to the embedded ARM processors. nn-X's measured performance was 200 G-ops/s, which is more than 83% of its theoretical peak performance.

The next application is the face detector used in [8]. We used a slightly modified version of this network. The first layer comprises 16 feature maps of 5×5 and is fully connected with the second convolution layer which comprises 64 feature maps of 7×7. Each of these layers is interspersed with max-pooling of 4×4 and thresholding. The input to the network was a 500×350 greyscale image. This network requires 552 M-ops per frame and includes a multi-scale pyramid with scales of 0.3, 0.24, 0.1. Construction of this pyramid is a pre-processing step that is performed on the ARM processors. The multi-scale input is then sent to the network for detection. nn-X was more than 115 times faster than the embedded ARM processors.

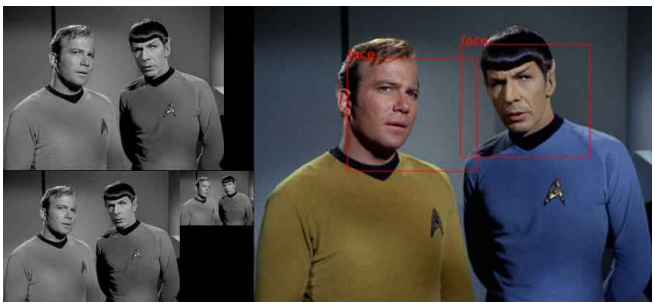

Figure 4: A face detector application with 552 M-ops per frame. nn-X was able to process a 500×350 video at 42 frames a second and was 115 times faster than the embedded processors. The image on the left is a multi-scale pyramid to provide scale-invariance to the input.

The third application was a street scene parser capable of categorizing each pixel of the input image into one of eight categories: buildings, cars, grass, persons, road, street signs, sky and trees. This network requires 350 M-ops to process one frame.

Figure 5 demonstrates nn-X performing full-scene understanding of a typical scene encountered when driving an automobile. nn-X processed a 510×288 video sequence in 4.5 ms, and was 112 times faster in processing time than the embedded ARM cores for this application.

We finally compared nn-X to other computing platforms commonly used to execute neural networks. The results (shown in Figure 6) report performance per unit electrical power consumed. Most desktop and laptop CPUs and GPUs

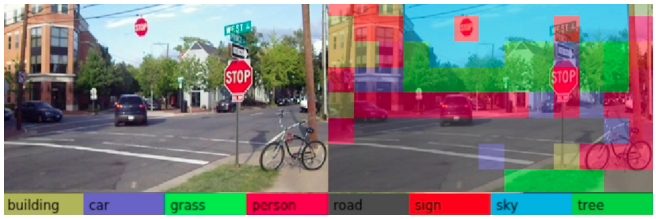

Figure 5: Road scene-parsing application requiring 350 M-ops per frame. This application processes a 510×288 video sequence in 4.5 ms and produces an 8-class label for each frame. nn-X is 112 times faster than the embedded processors.

peaked at under 3 G-ops/s-W even when the algorithm was optimized to take advantage of hardware acceleration. Mobile processors reported better efficiencies of 8 G-ops/s-W.

nn-X (red) implemented in programmable logic was able to deliver more than 25 G-ops/s-W. nn-X's embeddable factor is six times that of the Snapdragon 800 SoC and twenty times that of NVIDIA's GTX 780. Figure 6 compares nn-X to custom processors running at much higher frequencies. An implementation of nn-X in silicon at similar process nodes would significantly improve its performance.

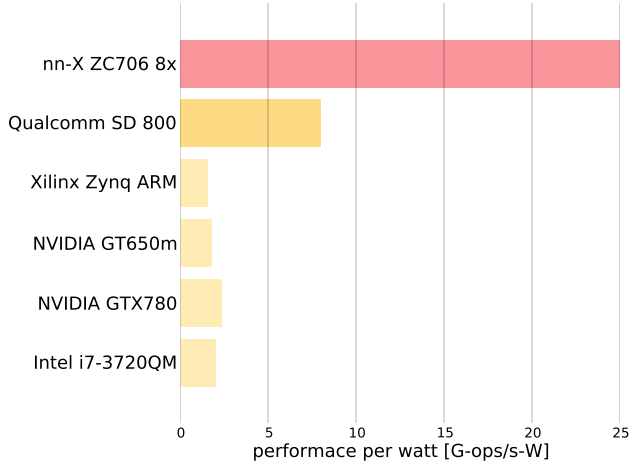

Figure 6: Performance per watt of different platforms. Most desktop CPUs and GPUs gave under 3 G-ops/s-W while mobile processors performed slightly better. nn-X (red) implemented in programmable logic was able to deliver more than 25 G-ops/s-W.

V. Discussion

In this section we analyze the large performance benefit of this architecture. One advantage is nn-X's large parallelism; eight convolutional engines of 10×10 can deliver up to 227 G-ops/s.

Another advantage is its efficient hardware utilization. The use of streaming data and lack of conditional statements allows nn-X to use every clock cycle to perform the desired operations. CPUs and GPUs are not able to use every clock cycle for useful computation. Conditional statements result in inefficient use of a processor's pipeline [14]. Another inefficiency arises from the long latency of main memory accesses. Processor caches are generally small compared to the total size of the inputs, intermediates and outputs [4]. This requires a processor to initiate memory accesses for data that is not cached.

In deep neural network applications, the convolution operator has few branches in its code as compared to the max-pooling and thresholding operators. Furthermore, branches in the convolution operator are highly predictable. Conditional statements in the max-pooling and thresholding operator, on the other hand, are difficult to predict because their path is based on the value of the input pixel. This causes a performance drop in CPUs due to branch mispredictions. On GPUs, control divergence causes a drop in performance throughput[12].

To demonstrate this, we used two model deep networks. The first model consisted of an input layer of 3×16 kernels of 10×10 and an output layer of 16×32 kernels of 7×7. The second model consisted of the same convolution layers but these were interspersed with max-pooling and thresholding operations. We used the same platforms from Figure 6 to perform this experiment.

In this context, we define efficiency as the performance achieved when running the model with only convolution layers versus the performance achieved when running the model with the max-pooling and threshold operations included. With the Torch7 package, all general purpose processors achieved an efficiency between 75% to 85%. nn-X achieved an efficiency close to 100%.

We explain this by the fact that in nn-X, the output of a convolution does not need to be written to memory due to the cascade of pipelined operators. Furthermore, as nn-X has no control flow, the output latency of the entire operation is simply equal to the combined latencies of each individual operator.

VI. Conclusion

We presented nn-X, a coprocessor for accelerating deep neural networks. Deep networks are used in synthetic vision systems because of their versatility and as such, are suitable for a variety of vision tasks. Deep networks like convolutional neural networks are inherently parallel and can be accelerated on custom hardware to give a low powered mobile system capable of achieving high performance.

We demonstrated the performance of nn-X on a single-layer neural network, a face detection and a road scene-understanding application. nn-X was faster than embedded processors in all these applications. More prominently, nn-X achieved better performance per watt than platforms that are commonly used to process deep networks. This makes

nn-X an excellent candidate as an embedded deep network accelerator and for mobile platforms.

ACKNOWLEDGMENT

Work supported by Office of Naval Research (ONR) grants 14PR02106-01 P00004 and MURI N000141010278.

REFERENCES

[1] L. Camunas-Mesa, A. Acosta-Jimenez, C. Zamarrefio-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco. A 32x32 pixel convolution processor chip for address event vision sensors with 155 ns event latency and 20 meps throughput. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 58(4):777–790, April 2011.

[2] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 247–257, New York, NY, USA, 2010. ACM.

[3] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 269–284, New York, NY, USA, 2014. ACM.

[4] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220, 2010.

[5] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[6] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 257–260, 2010.

[7] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, page 109116, 2011.

[8] C. Farabet, C. Poulet, and Y. LeCun. An fpga-based stream processor for embedded real-time vision with convolutional networks. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 878–885, Sept 2009.

[9] H. P. Graf, S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, and S. Chakradhar. A massively parallel digital learning processor. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 529–536, 2009.

[10] J. Holi and J.-N. Hwang. Finite precision error analysis of neural network hardware implementations. *Computers, IEEE Transactions on*, 42(3):281–290, Mar 1993.

[11] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[12] G. Li, P. Li, G. Sawaya, G. Gopalakrishnan, I. Ghosh, and S. P. Rajan. Gklee: Concolic verification and test generation for gpus. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 215–224, New York, NY, USA, 2012. ACM.

[13] A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14–22, Jan 2012.

[14] R. Nath, S. Tomov, and J. Dongarra. Accelerating gpu kernels for dense linear algebra. In J. Palma, M. Dayd, O. Marques, and J. Lopes, editors, *High Performance Computing for Computational Science VECPAR 2010*, volume 6449 of *Lecture Notes in Computer Science*, pages 83–92. Springer Berlin Heidelberg, 2011.

[15] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. Graf. A massively parallel coprocessor for convolutional neural networks. In *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, pages 53–60, 2009.

[16] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recognition with cortex-like mechanisms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):411–426, 2007.