An Efficient Hardware Implementation of Artificial Neural Network based on Stochastic Computing

Duy-Anh Nguyen, Huy-Hung Ho, Duy-Hieu Bui, Xuan-Tu Tran
SISLAB, VNU University of Engineering and Technology – 144 Xuan Thuy road, Cau Giay, Hanoi, Vietnam
danguyen@vnu.edu.vn; hhhung96@gmail.com; hieubd@vnu.edu.vn; tutx@vnu.edu.vn

***Abstract*—Recently, Artificial Neural Network (ANN) has emerged as the main driving force behind the rapid developments of many applications. Although ANN provides high computing capabilities, its prohibitive computational complexity, together with the large area footprints of ANN hardware implementations, has made it unsuitable for embedded applications with real-time constraints. Stochastic Computing (SC), an unconventional computing technique which could offer low-power and area-efficient hardware implementations, has shown promising results when applied to ANN hardware circuits. In this paper, efficient hardware implementations of ANN with conventional binary radix computation and SC technique are proposed. The system's performance is benchmarked with a handwritten digit recognition application. Simulation results show that, on the MNIST dataset, the 10-bit binary implementation of the system only incurs an accuracy loss of 0.44% compared to the software simulations. The preliminary simulation results of the SC neuron block show that the output is comparable to the binary radix results. FPGA implementation of the SC neuron block has shown a reduction of 67% in the number of LUTs slice.**

***Index Terms*—Artificial Neural Network, Stochastic Computing, MNIST data set**

I. Introduction

Artificial Neural Network (ANN) is currently the main driving force behind the development of many modern Artificial Intelligence (AI) applications. Some popular variants of ANN such as Deep Neural Network (DNN) and Convolutional Neural Network (CNN) have made some major breakthroughs in the field of self-driving car [1], cancer detection [2], or playing the game of GO [3]. Spiking Neural Network (SNN) model, the third generation of ANN, has been extensively used to simulate the human's brain and to study its behavior [4]. Despite having state-of-the-art performance, ANN also requires a very high level of computational complexity. This has made ANN unsuitable for many embedded image and video processing applications, e.g smart surveillance camera systems, portable voice recognition systems, etc. Those applications impose real-time processing and low-power consumption constraints. To enable real-time, in-field processing of ANN in such platforms, developing dedicated hardware architectures has attracted growing interest [5][6][7]. Such dedicated platforms have two major advantages. The first one is the high computational speed due to its innate parallel processing capabilities. The second one is the low power and low area cost due to the highly optimized architectures of the dedicated platform. In this paper, a hardware architecture for efficient processing of ANN is proposed, with major goals of achieving low area cost and moderate power consumption.

To further optimize the area cost and power consumption of the architecture, Stochastic Computing (SC) technique is also introduced. SC is an unconventional computing technique which was first proposed in the 1960s by Gaines [8]. SC allows the implementation of complex arithmetic operations with simple logic elements. Hence SC could greatly reduce the area cost of hardware architectures. Recently, SC has been successfully applied to many applications, such as edge-detecting circuit [9] or LDPC decoder circuit [10]. However, SC suffers from low computational accuracy issues, due to the random nature of the computation in SC domain [8]. On the other hand, ANN has inherent fault-tolerant nature and could show good results even with very low precision computing [11]. Therefore combining dedicated ANN hardware architecture with SC technique is a solution to reduce the area cost and power consumption of the system.

The two main contributions of the paper are as follows. Firstly, a dedicated hardware architecture for efficient processing of ANN is proposed. Secondly, SC computing technique is introduced to the existing platform to reduce the area cost. Simulation results

978-1-5386-7983-8/18/$31.00 ©2018 IEEE

show that on the MNIST dataset, the 10-bit fixed point implementation of the platform only incurs an accuracy loss of 0.44% compared to the software simulation results. The preliminary results of the system with SC technique applied show that the output is comparable to the base system. The FPGA implementation of the SC neuron block also shows a great reduction of 67% in the number of LUTs slice used.

The outline of the paper is as follows. Section II presents the basic concepts of SC and its computational elements. Section III covers the propose hardware architecture of the system in details. Section IV reports the simulation and implementation results on FPGA platform. Finally, Section V concludes the paper.

II. Basic of Stochastic Computing and its computational elements

A. Basic concept

Stochastic Computing is an unconventional computing method based on the probability of bits in a pseudo-random bit stream. Numbers in SC domain are represented as pseudo-random bit streams, called *Stochastic Numbers*, which can be processed by very simple logic circuits. The value of each stochastic number is determined by the probabilities of observing bit 0 and 1 in each bit stream. For example, let a pseudo-random bit stream S with a length of N denotes the stochastic number X. S contains N_1 1's and $N-N_1$ 0's . In the *unipolar* format, the probability p_x of X has the range of $[0;1]$, and is given as:

$$p_x = \frac{N_1}{N} \tag{1}$$

The stochastic representation of a number is not unique. For each number N_1/N, there are $\binom{N}{N_1}$ representations. For example, with $N = 6$, there are 6 ways to represent the number $5/6$: $\{111110, 111101, 111011, 110111, 101111, 011111\}$. With a given bit stream length N, there is only a small subset of real numbers in $[0;1]$ can be expressed exactly in SC.

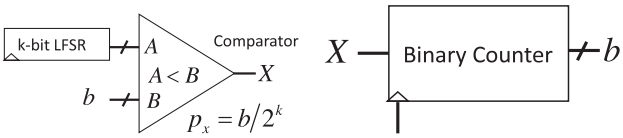
To express real numbers in different intervals, several other SNs format has been proposed [12]. One popular variant is the *bipolar* format. A stochastic number X in unipolar format with value $p_x \in [0;1]$ can be mapped to the range $[-1;1]$ via a mapping function:

$$p_y = 2p_x - 1 = \frac{N_1 - N_0}{N} \tag{2}$$

B. SC computational elements

Stochastic Computing uses a fundamentally different computing paradigm from the traditional method. SC requires some basic computational elements. The details are given in this section.

1) Stochastic Number converter: Circuits that act as interfaces between binary numbers to stochastic number are fundamental elements of SC. A circuit which converts a binary number to SC format is called a Stochastic Number Generator (SNG). On the other hand, a circuit that converts an SC number to binary number format is called a Stochastic Number Decoder. Fig. 1 shows such circuits.



(a) Stochastic Number Generator (b) Stochastic Number Decoder

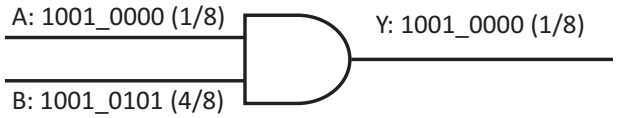
Figure 1: Circuits to convert between stochastic numbers and binary numbers.

An SNG consists of a Linear Feedback Shift Register (LFSR) and a comparator. A k-bit pseudo-random binary number is generated in each clock cycle by the LFSR, then it is compared to the k-bit input binary number b. The comparator produces '1' if the random number is less than b and '0' otherwise. Assuming that the input random numbers are uniformly distributed over the interval $[0;1]$, the probability of '1' appearing at the output of the comparator at each clock cycle is $p_x = b/2^k$. On the other hand, a stochastic number decoder consists of a binary counter. It will simply count the number of bit '1' in the input stochastic number X.

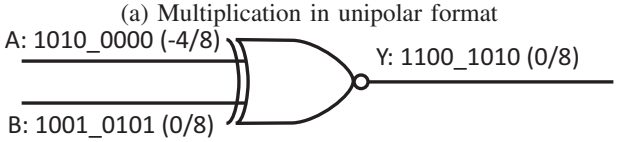
2) Multiplication in SC: Multiplication of two stochastic bit streams is performed using AND and XNOR gates in unipolar and bipolar format, respectively. Fig. 2a, 2b demonstrates examples of multiplication in SC domain.

3) Addition in SC: The addition in SC is usually performed by using either scaled adders or OR gates. Fig. 2c, 2d show examples of addition in SC using either a MUX or an OR gate.

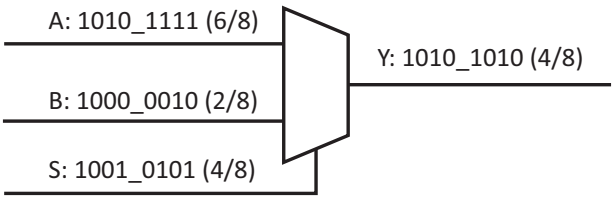
The addition in SC is also performed by using a stochastic adder that is more accurate and does not require additional random inputs like the scaled adder. This adder is proposed by V.T. Lee [13], and is illustrated in Fig. 3.



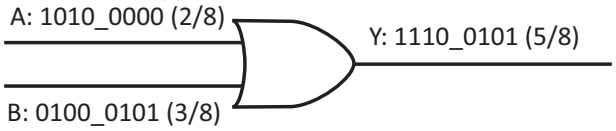
(a) Multiplication in unipolar format



(b) Multiplication in bipolar format



(c) Scaled adder with a MUX



(d) Addition using OR gate

Figure 2: Multiplication examples in SC.

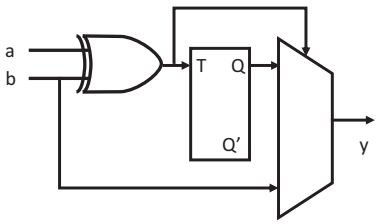


Figure 3: TFF-based stochastic adder.

The accuracy of operations in SC domain always depends on the correlation between the input bit streams. One advantage of this adder is that the bit-stream generated by the T Flip-flop (TFF) is always uncorrelated with its input bit stream. Moreover, the area cost of a TFF is no more than an LFSR that is required to generate the scaled value of $1/2$. Hence, this adder is adopted in this work.

III. Hardware Architecture for Efficient Processing of ANN

In this section, the proposed hardware architecture is discussed in details. The architecture aimed at efficiently processing the inference phase of multilayer, feed-forward ANN, also called Multi-Layer Perceptron (MLP). The training phase is conducted off-line.

A. The proposed architecture

The top architecture is shown in Fig.4.

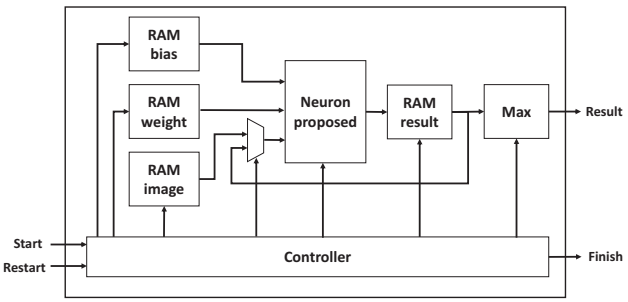


Figure 4: The proposed architecture.

At the beginning of each operation cycle, the input data, the trained connection weights and the biases are preloaded to the *RAM image*, *RAM weight*, and *RAM bias* blocks, respectively. The *Controller* block will then issue the *Start* signal to begin the calculation of the *Neuron proposed* block. The data from the input block RAM is processed at this block. The output of each neuron layer is stored at the *RAM result* block. A multiplexer is used to select the inputs for the *Neuron proposed* block, either from the *RAM image* for the first hidden layer, or the feedback from *RAM result* for each subsequent layer. The *Max* block will determine the classification results from the outputs of final neuron layer.

B. The binary neuron block

The architecture of the binary radix design is shown in Fig. 5. The architecture uses M parallel inputs x,

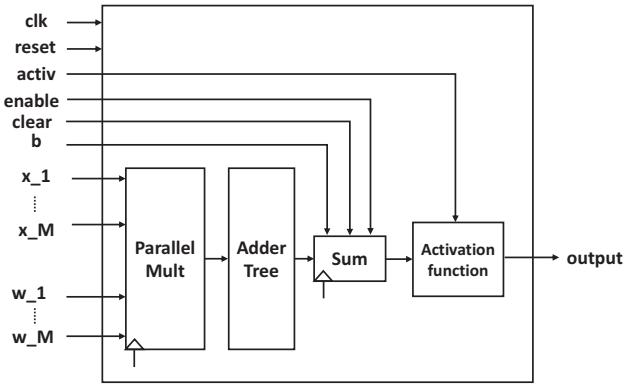


Figure 5: The binary neuron block.

M weight inputs w and bias b. The choice of M is configurable. Other inputs include the control signal from the *Controller* block. The *Parallel Mult* and the *Adder Tree* block will calculate the sum-of-product

between x and M. The output is loaded to *Sum* register. The *Activation function* block is used to implement the activation function of each neuron. In this work, two of the most popular activation functions are implemented, which are the Sigmoid function and the ReLU function. [14]

C. The SC neuron block

The binary neuron block uses a lot of parallel multipliers and adders, which are costly in terms of area cost. On the other hand, the multiplication and addition in SC can be implemented with area-efficient basic components. Hence, a novel neuron block based on SC technique is proposed in this section to reduce the area footprint. The proposed SC neuron block still maximize parallelism with M inputs. Because our application requires negative weights and biases, SC with bipolar format representation is chosen in this work. The architecture of the proposed SC neuron block is given in Fig. 6.

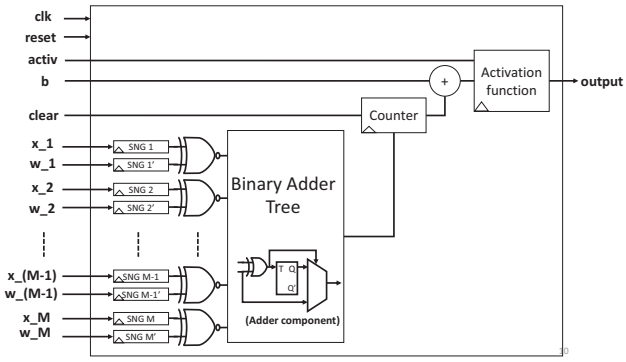


Figure 6: The SC neuron block.

The operation of the SC neuron block is as follows:

- At the start of each cycle, *clear* signal is activated, all SNGs and the counter are reset.
- The SNGs will generate SNs corresponding to M inputs (x_1 to x_M). and M weights (w_1 to w_M).
- The multiplication and addition of SNs will be realized by the XNOR gates and the binary adder tree with the efficient adder component as mentioned in section II. The output of the adder tree will be converted to binary radix through the counter.
- The bias is then added to the output of the counter. The activation function block will calculate the final output of the neuron.

IV. Simulation and Implementation Results

A. Software simulation results

The handwritten digit recognition application is chosen as a benchmark for our system performance. One of the most popular data set for this task is the MNIST dataset [15]. This dataset contains 60,000 images for training purpose and 10,000 images for testing purpose. The size of each image is 28×28.

The software implementation of ANN is realized with Caffe [16], a popular open-source platform for machine learning tasks. The training phase of ANN is performed off-line. The trained connection weights and biases are then used for the inference phase of our hardware system. The chosen ANN's model for our application is a Multi-layer perceptron with only 1 hidden layer. Since each input image has a size of 28×28, there are 784 neurons in the input layer. The classification digits are from 0 to 9, hence there are 10 neurons in the output layer. In this work, we also explore the effect of varying the number of neurons in the hidden layer on classification accuracy. Table I summarizes the classification results on Caffe platform. N is the number of neurons in the hidden layer. The chosen activation functions are the Sigmoid and ReLU functions. The classification results on software platform serve as a golden model to benchmark our hardware implementation results.

Table I: Classification accuracies with Caffe

N	16	48	64	800
Sigmoid	88.04%	89.01%	89.10%	89.48%
ReLU	92.17%	92.63%	92.91%	93.49%

It is clear that the classification accuracy is increasing with N. However, the huge network's size quickly becomes a constraint for efficient hardware implementations. On the other hand, with substantially lower N, the accuracy decreases by a small margin. For example, with ReLU activation function, when N decreases from 800 to 64, there is only a 0.58% accuracy loss. Thus in this work, we will limit the choice of N to 16-48-64 in our hardware implementations.

B. Results of ANN hardware implementations

The proposed design is realized with VHDL at RTL level. Table II and Fig. 7 show the classification results with the proposed binary radix hardware implementations. We used 10-bit fixed point representation in our design.

It can be seen that the accuracy's of our hardware implementation is comparable to the results from software simulations. With $N = 64$ and ReLU activation function, the best classification result is 92.47% with only a 0.44% accuracy loss. The accuracy loss

Table II: Comparison of classification accuracies between the proposed design and the Caffe's implementation

N		16	48	64
Sigmoid	Proposed design	87.60%	86.63%	85.94%
	Software simulation	88.04%	89.01%	89.10%
	Accuracy loss	0.44%	2.38%	3.16%
ReLU	Proposed design	91.56%	92.18%	92.47%
	Software simulation	92.17%	92.63%	92.91%
	Accuracy loss	0.61%	0.45%	0.44%

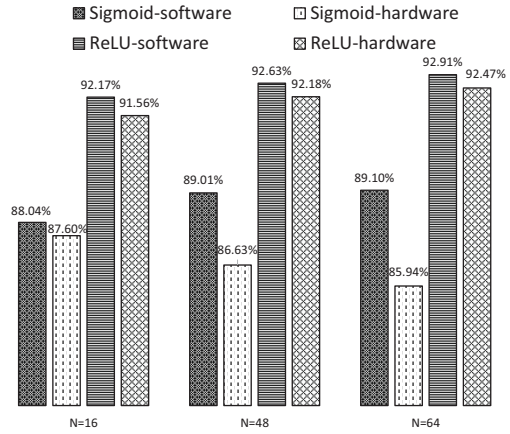


Figure 7: Classification results on software and hardware platform.

comes from the fact that our design uses 10-bit fixed point representation while Caffe's platform uses 32-bit floating point representation. Another observation is that the hardware implementation of Sigmoid incurs a greater accuracy loss compared to ReLU. This is because in the proposed design the Sigmoid function is approximated with a Look Up Table (LUT).

C. Results of SC neuron block implementations

To evaluate our SC neuron block implementation, the Mean Square Error (MSE) between the outputs of each block with SC technique and the outputs of SC's software implementation is used. MSE is calculated as [13]:

$$MSE = \sum_{i=0}^{N-1} \frac{(O_{floating\_point} - O_{proposed})^2}{N} \quad (3)$$

Table III compares the MSE of SC's multiplier and SC's adder between our implementation and the design in [13]. We used 8-bit fixed point representation in both SC's unipolar and bipolar format.

Table III: Comparison between the SC adder and SC multiplier's MSE with other works

	Design in [13]	Proposed design	
SC format	8-bit unipolar	8-bit unipolar	8-bit bipolar
Multiplier	2.57×10^{-4}	7.43×10^{-6}	1.98×10^{-4}
Adder	1.91×10^{-6}	1.89×10^{-6}	1.32×10^{-5}

This shows that the output of our basic SC's components is comparable to the one proposed by V.T. Lee in [13]. To further evaluate the performance of the proposed SC neuron block, we compare the output's MSE between the binary radix implementation and the SC implementation. The number of clock cycle to finish one classification is also included. The neuron block uses $M = 16$ parallel inputs. This is summarized in table IV:

Table IV: MSE and execution time comparison between binary radix neuron block and SC neuron block

	Bitwidth	Binary radix	SC
MSE	8	2.02×10^{-3}	7.05×10^{-2}
	10	1.47×10^{-4}	9.02×10^{-4}
Execution time (clock cycle)	8	2	257
	10	2	1025

The SC neuron's output's MSE is comparable to the binary radix implementation. However, SC technique results in longer latency.

D. Results of FPGA implementations

We have implemented the proposed binary radix design in a Xilinx Artix-7 FPGA platform. The number of hidden layer's neuron chosen is 48 with ReLU activation function and 10-bit fixed point representation. The results are summarized in Table V and Table VI.

Table V: Performance report of FPGA implementation

Performance metric	Result
Frequency	158 MHz
Clock cycle	4888
Execution time	30.93 μs
Total power	0.202 W

The maximum frequency is 158MHz. The execution time for one classification is 30.93 μs. The total power

Table VI: Hardware resources utilization report of FPGA implementation

Cost	Design	FPGA resources	Utilization
LUTs	1659	133800	1.24%
Register	1020	267600	0.38%
Mux	0	100350	0%
Slice	578	33450	1.72%
DSP	0	740	0%
BRAM	0.5	365	0.14%
IO	8	400	2.0%

consumption is 0.202 W. The area cost is also small compared to the available FPGA resources, with 1659 LUTs slices and 1020 register slices.

We have also implemented the SC neuron block in this FPGA platform. Table VII summarizes the area cost of SC neuron block and binary neuron block.

Table VII: FPGA implementation results of Binary neuron block and SC neuron block

	Binary neuron block	SC neuron block
Parallel inputs	16	16
Frequency	250 MHz	286 MHz
LUTs	1268	416
Register	23	299
IO	277	277
Power consumption	0.045W	0.039W

The SC neuron block's implementation results show that, with SC technique applied, the neuron block could have faster operation frequency, use fewer LUTs slice (67% reduction) and lower power consumption. However, the SC design uses more register slice due to the need for a large number of SNGs.

V. Conclusion

ANN has been the major driving force behind the developments of many applications. However, its high computational complexity has made it unsuitable for many embedded applications. In this work, we introduced an efficient hardware implementation of ANN, with SC technique applied to reduce the area cost and the power consumption of the design. The chosen ANN's network model is a feed-forward multi-layer perceptron for digit recognition application. The binary radix implementation of the design shows comparable results with the software implementation, with up to 92.18% accuracy. With SC technique applied, the neuron block has lower power consumption and lower numbers of LUTs slice.

References

[1] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 2722–2730.

[2] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, pp. 115–117, Jan 2017.

[3] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–485, Jan 2016.

[4] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov 2003.

[5] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "Cnp: An fpga-based processor for convolutional networks," in *2009 International Conference on Field Programmable Logic and Applications*, Aug 2009, pp. 32–37.

[6] 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.

[7] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *CVPR 2011 WORKSHOPS*, June 2011, pp. 109–116.

[8] B. R. Gaines, *Stochastic Computing Systems*. Boston, MA: Springer US, 1969, pp. 37–172.

[9] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–6.

[10] W. J. Gross, V. C. Gaudet, and A. Milner, "Stochastic implementation of ldpc decoders," in *Conference Record of the Thirty-Ninth Asilomar Conference onSignals, Systems and Computers, 2005.*, Oct 2005, pp. 713–717.

[11] B. Moons, B. D. Brabandere, L. V. Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2016, pp. 1–8.

[12] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 149–156.

[13] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 13–18.

[14] 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.

[15] "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[16] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.