A Convolutional Accelerator for Neural Networks With Binary Weights

Arash Ardakani, Carlo Condo and Warren J. Gross
Department of Electrical and Computer Engineering, McGill University, Montréal, Québec, Canada

***Abstract*—Parallel processors and GP-GPUs have been routinely used in the past to perform the computations of convolutional neural networks (CNNs). However, their large power consumption has pushed researchers towards application-specific integrated circuits and on-chip accelerators implement neural networks. Nevertheless, within the Internet of Things (IoT) scenario, even these accelerators fail to meet the power and latency constraints. To address this issue, binary-weight networks were introduced, where weights are constrained to -1 and 1. Therefore, these networks facilitate hardware implementation of neural networks by replacing multiply-and-accumulate units with simple accumulators, as well as reducing the weight storage. In this paper, we introduce a convolutional accelerator for binary-weight neural networks. The proposed architecture only consumes 128 mW at a frequency of 200 MHz and occupies 1.2 mm^2 when synthesized in TSMC 65 nm CMOS technology. Moreover, it achieves a high area-efficiency of 176 Gops/MGC and performance efficiency of 89%, outperforming the state-of-the-art architecture for binary-weight networks by 1.8$\times$ and 3.2$\times$, respectively.**

I. Introduction

Deep neural networks (DNNs) have been gaining popularity within different research fields due to their excellent capability in extracting features and processing raw data. In particular, the industry has recognized their crucial role in Internet of Things (IoT) frameworks. DNNs are in fact a effective tool to process sensory data collected from IoT devices connected to the internet with sensing and processing capabilities. In the past, high-performance platforms such as graphics processing units (GPUs) and central processing units (CPUs) were conventionally exploited to utilize DNN models for such applications. However, response time and power/energy consumption of these hardware platforms are higher than the budget of IoT devices, due to high complexity degree of DNN models. As a result, application-specific integrated circuits (ASICs) and dedicated accelerators for deep learning has become widespread. While these specialized hardware are faster and consume significantly less power than GPUs and CPUs, they still cannot meet the power and response time constraints of IoT systems.

To address the above issue, many approaches were introduced in literature. Spiking neural networks and stochastic models were considered as a solution for low-power implementations of DNNs in [1]–[3], thanks to their ultra low-cost hardware units. In these models, numbers are represented as sequence of random bits, allowing to perform computations using simple logic gates. For instance, multiplications are performed through XNOR gates in stochastic computing, which significantly reduces the power consumption of DNNs. It is worth mentioning that the stochastic models are also fault tolerant. Despite their advantages, the nature of stochastic computing leads to long latencies.

Pruning techniques were also introduced to reduce the number of parameters of DNNs and consequently the power dissipation of memory accesses to the off-chip memory [4], [5]. It is worth mentioning that higher classification accuracy can also be achieved using the pruning technique introduced in [5]. While these methods can speed up the computations on GPUs and CPUs, no practical speedup was observed for convolutional layers of DNNs on dedicated hardware.

Binary-weight networks were introduced in [6]. These networks are obtained by constraining weights to -1 and 1 during the training phase. Since these networks require a single bit to represent each weight, multiplications can be performed using low-cost multiplexers. The majority of current research efforts consider implementations of binary-weight networks. For instance, the convolutional accelerator presented in [7] for binary-weight networks shows significant reduction in energy/power consumption and silicon area compared to other implementations of DNNs that rely on 16 bits for both weights and pixels. However, the existing convolutional accelerators of binary-weight networks still suffer from low performance efficiency, where the performance efficiency is computed as the ratio between the measured and the peak number of operations per second. The performance efficiency in [7] is limited to 28% only. Therefore, an efficient implementation achieving a performance efficiency close to 100% is still missing.

In [8], we introduced a convolutional computation data flow and its corresponding hardware implementation, achieving performance efficiency close to 90%. In this paper, we propose a novel architecture exploiting this data flow to improve the performance efficiency of binary-weight networks, and show that it outperforms the existing architectures in literature in terms of both area and performance efficiency.

II. Preliminaries

A. Deep Neural Networks

DNNs consist of multiple layers of neurons between input and output layers, with weighted connections between neurons of different layers. DNNs identify two different phases: the training and inference phases. During training, the learning engine adjusts the weights associated to each connection, while the inference engine uses the extracted weights to classify, recognize or process data. Convolutional neural networks (CNNs) [9] are the most important class of DNNs, widely used to analyze visual imagery. CNNs are constructed by stacking convolutional layers followed by a few fully-connected layers. In [9], it was shown that the connectivity among neurons in convolutional layers follows a pattern similar to the organization of the visual cortex of animals, that can be mathematically expressed by convolution operations. Therefore, a set of weights is shared among all neurons. In fact, neuron

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

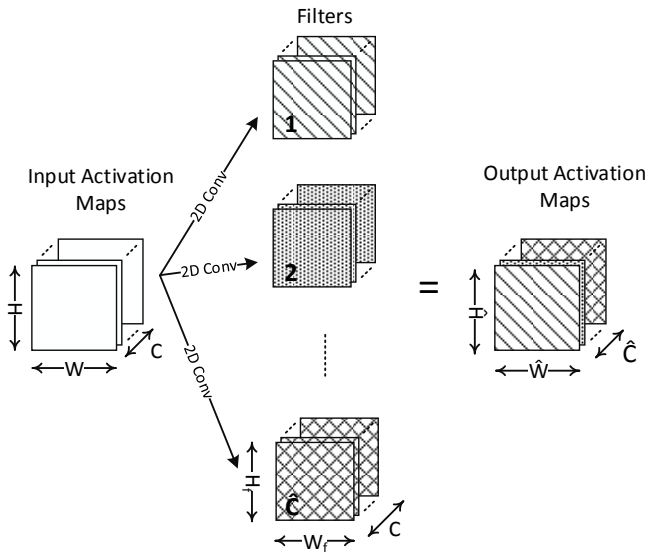

Fig. 1. The convolutional computations of each layer.

connectivity in convolutional layers is sparse, as opposed to fully-connected layers where each neuron is connected to every other neuron in the previous and next layers.

B. Convolutional Computations

Fig. 1 illustrates the computations of a single convolutional layer. A set of weights, which is hereafter referred to as a filter, is shared among all neurons in the layer. Neurons (i.e. pixels of activation maps) in convolutional layers are arranged in 3 dimensions: height, width and channel, and each neuron performs multiplications and accumulations. Each convolutional layer maps 3D input data (AKA 3D input activation maps) into 3D output activation maps as a result of the convolution between input activation maps and 4D $H_f \times W_f \times C \times \hat{C}$ filters. More precisely, each single 2D $\hat{H} \times \hat{W}$ plane of the output activation maps, that corresponds to a channel, is a convolution between 3D $H \times W \times C$ input activation maps and a set of 3D $H_f \times W_f \times C$ filters. Therefore, 3D convolution is a summation of multiple plane-wise 2D convolutions followed by addition with a 1D bias.

The aforementioned computations can be expressed as

$$y(z,t,q) = b(q) + \sum_{k=1}^{C}\sum_{j=1}^{H_f}\sum_{i=1}^{W_f} x(zS+j, tS+i, k) \times \\ \times w(j,i,k,q),$$

$$\hat{H} = (H - H_f + S)/S, \quad (1)$$

$$\hat{W} = (W - W_f + S)/S,$$

where $1 \le z \le \hat{H}$, $1 \le t \le \hat{W}$ and $1 \le q \le \hat{C}$. The input activation maps, the output activation maps, the filters and the bias matrices are denoted as x, y, w and b, respectively. The stride value S controls the shift amount after each convolution.

TABLE I. THE FULLY-CONNECTED INSPIRED DATA FLOW FOR CONVOLUTIONAL PROCESSES FOR $N = 6$.

	Tile					
Clock Cycles	Output #1	Output #2	Output #3	Output #4	Output #5	Output #6
Clk #1	$X_1 \times W_1$	$X_1 \times 0$	$X_1 \times 0$	$X_1 \times 0$	$X_1 \times 0$	$X_1 \times 0$
Clk #2	$X_2 \times W_2$	$X_2 \times W_1$	$X_2 \times 0$	$X_2 \times 0$	$X_2 \times 0$	$X_2 \times 0$
Clk #3	$X_3 \times W_3$	$X_3 \times W_2$	$X_3 \times W_1$	$X_3 \times 0$	$X_3 \times 0$	$X_3 \times 0$
Clk #4	$X_4 \times 0$	$X_4 \times W_3$	$X_4 \times W_2$	$X_4 \times W_1$	$X_4 \times 0$	$X_4 \times 0$
Clk #5	$X_5 \times 0$	$X_5 \times 0$	$X_5 \times W_3$	$X_5 \times W_2$	$X_5 \times W_1$	$X_5 \times 0$
Clk #6	$X_6 \times 0$	$X_6 \times 0$	$X_6 \times 0$	$X_6 \times W_3$	$X_6 \times W_2$	$X_6 \times W_1$
Clk #7	$X_7 \times 0$	$X_7 \times 0$	$X_7 \times 0$	$X_7 \times 0$	$X_7 \times W_3$	$X_7 \times W_2$
Clk #8	$X_8 \times 0$	$X_8 \times 0$	$X_8 \times 0$	$X_8 \times 0$	$X_8 \times 0$	$X_8 \times W_3$

C. VGGNets and VGG-like Networks

VGGNet [10] is a well-known CNN, containing 13 layers of convolutions and achieving 27% misclassification rate on the ImageNet dataset. Recently, VGG-like networks have also shown excellent performance when exploited in binary-weight networks [6]. VGGNets and VGG-like networks consist of convolutional layers in which the size of filters and strides are fixed to 3×3 and one pixel, respectively. The linear rectifier (ReLU) is used as a non-linear activation function for all layers. In this paper, we focus on the acceleration of the convolutions of these networks, due to their outstanding performance on a wide range of applications and datasets.

D. Fully-Connected Based Data Flow for Convolutional Processes

In [8], we introduced a new data flow inspired by the data flow of fully-connected computations to process convolutions. We also showed that a fully parallel implementation of the proposed data flow results in unacceptable area occupation and power consumption. To address this issue, we proposed the use of a subset of N neurons to process the convolutional computations. For instance, Table I shows the data flow for $N = 6$ when $H = 8$, $W = 8$, $C = 1$, $H_f = 1$, $W_f = 3$ and $\hat{C} = 1$. This data flow relies on 1D convolutions, since $H_f = 1$: therefore, 2D convolutions can be easily achieved by repeating the same procedure three times. Using the same concept, performing the 2D convolution $\hat{C}$ times leads to the 3D convolution required for convolutional layers. However, since only three neurons are active at each clock cycle (see Table I), we proposed the use of three neurons, instead of N neurons, to perform the convolutions of each 2D plane of the output activation maps (denoted using different colors in Table I). Each weight is sequentially input to the first neuron, and then shifted and passed to the second and third neurons.

III. PROPOSED ARCHITECTURE FOR BINARY-WEIGHT CONVOLUTIONAL LAYERS

Fig. 2 shows the proposed architecture for a binary-weight convolutional layer computations. This basic unit is referred to as a 1D tile. It consists of a weight generator and three neurons, each implemented with a processing elements (PE). The weight generator uses five registers to provide each neuron with an appropriate weight: the last two registers are used to handle the weight passing from a neuron of a row to a neuron of another row, as discussed in [8]. At first, the first three registers R_1, R_2 and R_3 are loaded with the first row of the first filter. Then, one clock cycle delay for output activation maps is provided by looping though them. Register R_4 and R_5 are used whenever the weight passing occurs.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

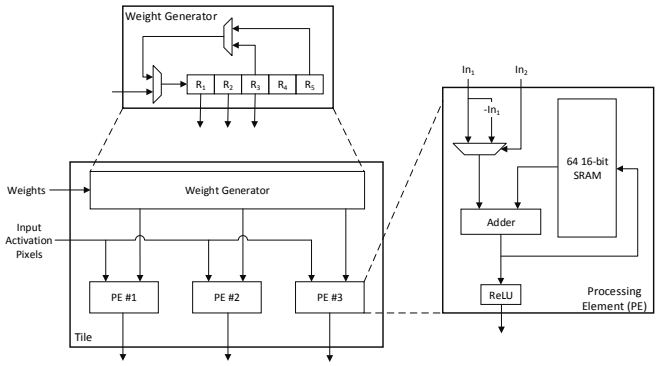

Fig. 2. Architecture of a single 1D tile.

Each PE takes as input an activation pixel and its corresponding weight generated by the weight generator unit. The multiplication-accumulation is performed using a simple multiplexer and adder thanks to the binary values of weights, as shown in Fig. 2. In the first three clock cycles, each PE performs the multiplication-accumulation for the first row of the first input filter and the computed partial value is stored in a memory. Using the same weights and the same PE, partial values for another output activation are computed and stored in the memory. Upon the completion of $N+2$ clock cycles, where N is the number of output activation pixels computed by each neuron, the computations of the second row of the first input filter start, and results are accumulated to the partial value of the first output activation pixels. The same procedure is also repeated for the third row of the first input filter. Therefore, the computations related to the first filter are completed in $3 \times (N+2)$ clock cycles. More generally, upon completion of $3 \times (N+2) \times C$ clock cycles, the output value of each PE is valid. Afterwards, each output value is passed to the ReLU and the result is stored on off-chip memories.

Performing the convolutional computations using the proposed architecture results in a high utilization factor (UF). In fact, the reutilization of the three neurons keeps them active for N out of $N+2$ clock cycles when computing each row of the input filters. Therefore, UF can be obtained by

$$\text{UF} = \frac{N}{N+2}. \tag{2}$$

This equation suggests a high performance efficiency of the proposed architecture for high values of N. It is worth noting that using higher values of N only increases size of the memories storing the partial values.

We also exploit p parallel tiles, where each is responsible for the output activation pixels of one or multiple output channel $\hat{C}$. Therefore, p times speed up in computations is expected when using p parallel tiles. Since the input pixels are shared among all p tiles, memory accesses are also reduced [8]. Fig. 3 shows the proposed accelerator for binary-weight networks. The total number of clock cycles (CCs) required for each VGG or VGG-like convolutional layer can be computed as follows:

$$\text{No. CCs} = 3 \times (N+2) \times C \times \frac{\hat{C}}{p} \times \frac{\hat{W} \times \hat{H}}{N}. \tag{3}$$

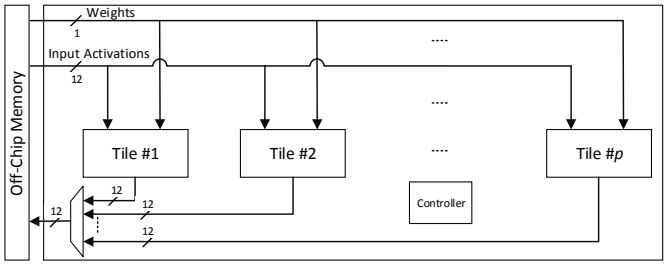

Fig. 3. Convolutional layer architecture with parallel tiles.

IV. Hardware Implementation Results

In [8], we showed that the choice of N and p leads to different types of implementations: from low-power to high-throughput systems. Similarly, the hardware performance of the proposed accelerator for binary-weight networks also depends on these two variables. As discussed in Section III, as the value of p increases the throughput and area occupation of the proposed architectures also linearly increase, while memory accesses to the off-chip memory decrease. On the other hand, a higher value of N reduces the off-chip memory accesses without any impact on the throughput of the system.

In this work, we optimized the proposed accelerator for a high-throughput, low-memory access implementation while keeping power consumption below the budget of IoT devices. The proposed accelerator contains $p = 64$ parallel tiles, in which each tile works based on $N = 192$ at a frequency of 200 MHz. Every tile contains three neurons, where each is associated with a 64-16b (i.e. 128-Byte) memory to store the partial product values. The input and output activation pixels are truncated to 12 bits, and the weights are represented using a single bit where 0 and 1 denote the values of -1 and 1, respectively. Our simulation results shows that this fixed-point configuration does not incur performance degradation with respect to the binary-weight networks presented in [5]. The tile pipelining technique introduced in [8] was also used to reduced the off-chip memory bandwidth.

Table II shows implementation results of the proposed hardware accelerator for binary-weight networks compared to other existing custom accelerators. The proposed architecture was fully tested on CIFAR10 [13] datasets using the model presented in [5], [6]. The proposed architecture contains 192 MAC units and achieves a real peak performance of 76 Gops at 200 MHz, where the real peak performance is defined as the maximum achievable performance that an architecture can get under certain conditions such as the layer size. Considering each multiplication-accumulation as two operations, the theoretical peak performance of any architecture can be computed as

$$\text{peak performance} = 2 \times \#\text{MAC} \times f, \tag{4}$$

where #MAC and f denote the total number of MAC units and the nominal frequency, respectively. Of course the real peak performance of accelerators might differ from the theoretical one. More precisely, the theoretical peak performance ignores the pre-processing and post-processing steps such as reading and writing data, as opposed to the real peak performance

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

TABLE II. HARDWARE IMPLEMENTATION RESULTS OF THE PROPOSED CONVOLUTIONAL ACCELERATOR FOR BINARY-WEIGHT NETWORKS

Reference	DATE'17 [11]	JSSC'17 [12]		TCAS'17 [8]		VLSI'16 [7]	This work
Technology	28 nm	65 nm		65 nm		65 nm	65 nm
Type	NA	NA		Area-efficient	MA-efficient	NA	NA
Gate Count (GC)[1](NAND-2)	3751k	1852k		565k	1117k	2160k	383k
Core Area[1](mm^2)	NA	12.52		1.77	3.5	NA	1.2
# MAC	576	168		96	192	1024	192
On-chip SRAM (kB)	352	181.5		43	86	NA	18.4
Nominal Frequency (MHz)	700	250		400	200	400	200
Peak Performance (Gops)	806.4	84		76	76	755	76
Bitwidth (bits)	16 fixed	16 fixed		16 fixed		12 fixed/binary	12 fixed/binary
Filter-sizes	NA	1-12 (h), 1-32(v)		3×3		NA	3×3
Channels	NA	1-1024		all		NA	all
Filters	NA	1-1024		all		NA	all
Layers	NA	all		all		NA	all
Dataset	ImageNet	ImageNet	ImageNet	ImageNet		CIFAR-10	CIFAR-10
CNN type	AlexNet	AlexNet	VGG-16	VGG-16		VGG-like	VGG-like
Voltage (V)	0.9	1	1	1	1	1.2	1
Power[1](mW)	567.5	278	236	254	260	153	128
Total Latency (ms)	393.36	115.3	4309.5	436.4	453.3	5.73	18
Throughput (fps)	326.2	34.7	0.7	2.29	2.21	175	55.5
Performance (Gops)	433.9	46.1	21.4	70.3	67.7	212	67.4
Performance Efficiency	54%	55%	26%	93%	89%	28%	89%
Energy-Efficiency[1](Gops/W)	764.6	166	90.7	276.8	260.4	1384	526.6
Area-Efficiency[1][2](Gops/MGC)	116	25	12	124	61	98	176

[1] Including on-chip SRAM.
[2] MGC stands for Mega Gate Count.

that considers all the required steps. The implementation results in Table II shows that the proposed accelerator process the convolutional computations of CIFAR-10 dataset within 18 ms, leading to a power consumption of 128 mW and 526.6 Gops/W, with a performance efficiency of 89%. It also occupies 1.2 mm^2 in a TSMC 65 nm CMOS technology. In fact, the proposed architecture meets all the implementation constraints (i.e., the power consumption and response time constraints) of IoT systems in which the power consumption and response time are limited to a few hundred mW [7] and a few milliseconds [14], respectively.

Recently, many convolutional accelerators have been published in literature [8], [11], [12]. However, they mainly focused on hardware implementations of fixed-point models, resulting in large power consumptions. In [7], the first ASIC implementation of binary-weight networks, known as YodaNN, was introduced. YodaNN was implemented in a UMC 65 nm CMOS technology, containing 1024 MAC units and working at nominal frequency of 400 MHz. According to (4), YodaNN achieves a theoretical peak throughput of 819.2 Gops and a real peak performance of 755 Gops. However, its measured performance is only 212 Gops when performing the convolutional computations of CIFAR-10, leading to a low performance efficiency of 28%, against the 89% efficiency achieved by our architecture (3.2× improvement). As a result, despite of its high energy-efficiency, YodaNN suffers from a low area-efficiency of 98 Gops/MGC while the proposed accelerator achieves an area-efficiency of 176 Gops/MGC (1.8× improvement).

V. CONCLUSION

In this paper, we proposed a convolutional accelerator for binary-weight networks. To this end, we used a data flow inspired from the computations of fully-connected layers. The proposed architecture can achieve a higher energy-efficiency when compared to accelerators based on fixed-point models. The proposed accelerator achieves a high area-efficiency of 176 Gops/MGC and performance efficiency of 89%, outperforming the state-of-the-art architecture for binary-weight networks by 1.8× and 3.2×, respectively. Moreover, the proposed architecture only consumes 128 mW, making it suitable to be exploited in IoT devices.

REFERENCES

[1] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "TrueNorth: design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct 2015.

[2] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2688–2699, Oct 2017.

[3] S. C. Smithson, K. Boga, A. Ardakani, B. H. Meyer, and W. J. Gross, "Stochastic Computing Can Improve Upon Digital Spiking Neural Networks," in *2016 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2016, to appear, pp. 1–6.

[4] 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.

[5] A. Ardakani, C. Condo, and W. J. Gross, "Sparsely-Connected Neural Networks: Towards Efficient VLSI Implementation of Deep Neural Networks," *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Nov. 2016.

[6] M. Courbariaux, Y. Bengio, and J. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *CoRR*, vol. abs/1511.00363, 2015.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

[7] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 236–241.

[8] A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, "An Architecture to Accelerate Convolution in Deep Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Early Access, doi: 10.1109/TCSI.2017.2757036, 2017.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[10] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014.

[11] S. Wang, D. Zhou, X. Han, and T. Yoshimura, "Chain-NN: An Energy-Efficient 1D Chain Architecture for Accelerating Deep Convolutional Neural Networks," *arXiv preprint arXiv:1703.01457*, 2017.

[12] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan 2017.

[13] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[14] T. Y.-H. Chen, L. S. Ravindranath, S. Deng, P. V. Bahl, and H. Balakrishnan, "Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices," in *13th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Seoul, South Korea, November 2015.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE