Theme Article

Image Recognition Accelerator Design Using In-Memory Processing

Yeseong Kim, Mohsen Imani, and Tajana Simunic Rosing
University of California San Diego

***Abstract*—This paper proposes a hardware accelerator design, called object recognition and classification hardware accelerator on resistive devices, which processes object recognition tasks inside emerging nonvolatile memory. The in-memory processing dramatically lowers the overhead of data movement, improving overall system efficiency. The proposed design accelerates key subtasks of image recognition, including text, face, pedestrian, and vehicle recognition. The evaluation shows significant improvements on performance and energy efficiency as compared to state-of-the-art processors and accelerators.**

Visual object recognition techniques are widely used to analyze images and videos. For example, autonomous vehicles need to recognize various objects such as cars, pedestrian, and traffic signals. Since most object recognition procedures are both data and compute intensive, general-purpose processors often do not offer enough power efficiency and performance to meet requirements of real-time responses.

In this paper, we present a novel accelerator design, called object recognition and classification hardware accelerator on resistive devices (ORCHARD), which performs the object recognition computation inside memory. Our design utilizes zero leakage power and fast read operations of memristor technology while reducing the significant data movement costs between processors and memory. The proposed accelerator consists of computation-enabled memory blocks that store the image data and perform machine learning tasks for image recognition. The in-memory processing design performs both the feature extraction procedure and image classification tasks, which are key components of the object recognition. It supports two popular feature extraction algorithms, histogram of oriented gradient (HOG) and Haar-like feature extraction.[1] It then performs the image classification tasks of the decision tree (DT) based ensemble algorithm which is one of the best image recognition methods.[2] In our evaluation, we show that the proposed ORCHARD design successfully performs

Digital Object Identifier 10.1109/MM.2018.2889402
Date of publication 3 January 2019; date of current version 21 February 2019.

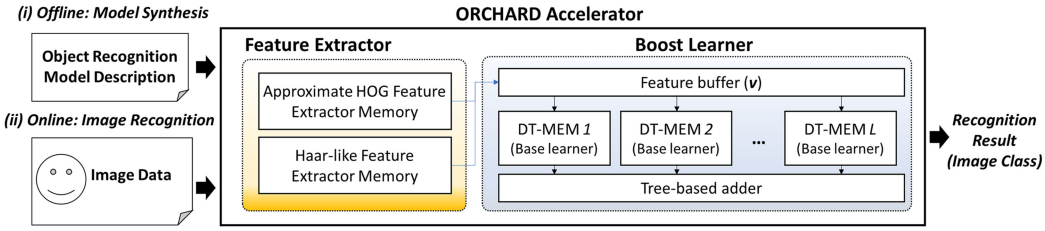

Figure 1. Overview of ORCHARD design.

four practical image recognition tasks: text, face, pedestrian, and vehicle recognition. Our design is 500× faster, and has 2100× higher energy efficiency with only 0.3% error as compared to the general-purpose processor-based computation.

ORCHARD DESIGN

Figure 1 shows an architectural overview of the proposed ORCHARD design. The object recognition models are developed offline, and written into the ORCHARD memory blocks. It then processes in-memory image recognition procedure at runtime. Our design supports modern feature extraction procedures using two memory-based modules, *approximate HOG feature extractor* and *Haar-like feature extractor module*. The extracted features are written into the feature buffer of the boost learner. The boost learner performs image classification tasks by running modern ensemble algorithms, e.g., cascade classifier, adaptive boosting and random forest, which use multiple DTs. We model the functionality of a DT and map that into a memory block, called *DT-MEM*. This memory block performs the decision steps based on in-memory search operations, while handling multiple images in a pipeline stage. Each DT-MEM produces a list of probability values for the recognized objects as the outputs. They are accumulated using a tree-based adder to create the final ensemble recognition result. Then, the image class with the highest probability sum is determined as the final recognition result. Since the major computation steps of the recognition procedure are implemented with in-memory computing operations, the ORCHARD design significantly reduces power and performance overhead due to data movement costs.

Memory-Based Feature Extraction

Approximate HOG feature extraction. Figure 2 illustrates the HOG feature extraction procedure with an example. The procedure first divides the original 32 × 32 image into four regions. For each region, it computes the gradient values of all pixels by considering its adjacent pixels (cell). The gradient can be represented by a vector which has an orientation (direction) and magnitude. The magnitude values of all cells are then accumulated into evenly-spread histogram bins. In this example, if a vector has a magnitude of m with an orientation of 230°, considering 8 bins from 0° to 315°, say v_i $(0 \leq i < 8)$, m is accumulated to v_5. This procedure computes the histogram bins for other regions as well, producing $R \cdot B$ features where R is the number of regions and B is the number of bins. If the image includes multiple color channels, applications may extract features from each color channel separately and combine all to a single feature vector. The main bottleneck of this procedure is the vector computation, as it typically involves many arithmetic operations, e.g., gradient calculation and trigonometrical functions.

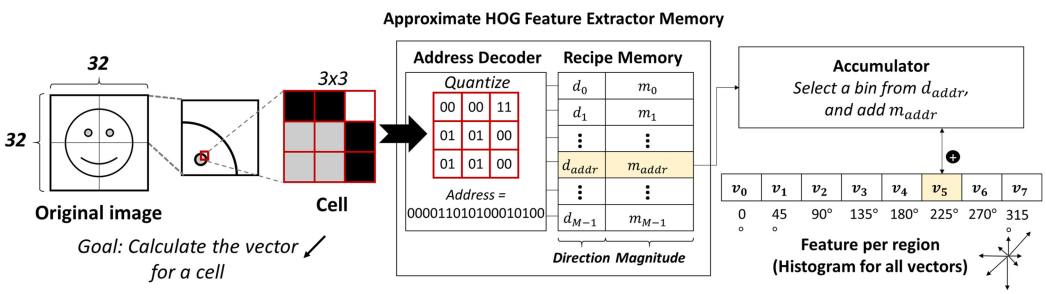

Figure 2. In-memory HOG feature extraction.

We optimize the vector computation by modeling and approximating it as a single memory access. For example, if the goal is to detect a handwritten alphabet,

Published by the IEEE Computer Society

we would approximate the input pixels using two levels, e.g., black and white, respectively. In that case, a cell includes 9 pixels, and thus the possible number of gradient values are only 2^9 (=512). Thus, we can precompute all gradient values and store into the cost-effective nonvolatile memory (NVM) blocks, so that it is directly fetched in the future computation. In this example, the 256 possible values are quantized to four levels, 00, 01, 10, and 11. The quantized values are concatenated to form a memory address that indicates a row of the crossbar memory block, called *recipe memory*. Each row of the recipe memory includes two pieces of information, one for the bin index of the vector direction d_i and the other for the magnitude m_i. The ORCHARD reads the gradient value from the memory row and accumulates it with small CMOS-based logic to compute the histogram of the region.

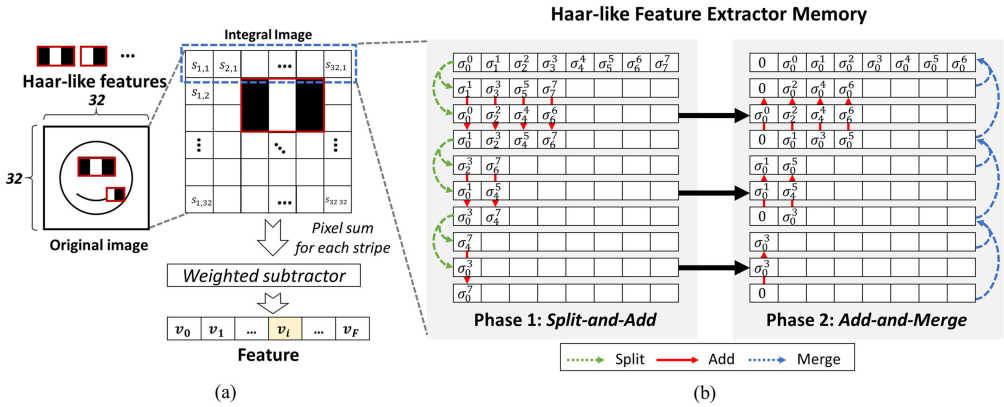

Figure 3. In-memory Haar-like feature extraction. (a) Flow of Haar-like Feature Extraction. (b) Integral Image Computation.

HAAR-LIKE FEATURE EXTRACTION. ORCHARD also supports Haar-like feature extraction procedure. Figure 3(a) describes the original procedure. This figure shows two types of Haar-like features denoted by the red boxes which are divided by black and white stripes. For example, the feature consists of two black stripes and one white stripe can capture the property that the color of eyes is different from the facial color. Since computing a sum of pixels in a stripe directly from the original pixels is cost ineffective, the state-of-art algorithm utilizes the concept of the integral image.[3] The integral image has the same size to the original images. Let us assume that each pixel of the original image is $p(x,y)$. A value of an integral image is $s(x,y) = p(x,y) + s(x-1,y) + s(x,y-1) + s(x-1,y-1)$, where $s(0,0)$ is zero. The element of the integral image $s(x,y)$ represents the sum of all pixels of a rectangle whose top-left coordinate is (1,1) and bottom-left coordinate is (x,y). Based on the integral image, the pixel sum of a stripe can be calculated as $(s(A) + s(D)) - (s(B) + s(C))$, where A, B, C, and D are the top-left, top-right, bottom-left, and bottom-right coordinates, respectively.

We parallelize the integral image computation based on an in-memory addition operation used in our previous work.[4] This operation adds all values of different columns utilizing in-memory NOR operations. This operation can be implemented on any resistive devices without using any CMOS gates. We compute the integral image by calculating the *prefix sum*[3] for image rows and columns. Let us assume that the pixels in a row are $p_0, p_1, \ldots, p_{1024}$ and $\delta_a^b = \sum_{i=a}^{b} p_i$. The prefix sum of the row is defined as $0, \delta_0^0, \delta_0^1, \ldots, \delta_0^{1023}$. Once computing the prefix sum of all rows, we take the transpose of the image, and run the same algorithm to obtain the integral image.

Figure 3(b) depicts an example of how ORCHARD computes the prefix sum of eight values in parallel, where the first memory row includes the pixel values of an image row. This procedure consists of two phases: 1) building a binary tree structure that has *partial sums* for each tree node; and 2) sweeping the partial sums to get the prefix sum. In the first phase, we copy the first memory row into two next rows, so that values at even columns and odd columns are split. Then, we perform the column-parallel in-memory addition, and the partial sum results are written into the fourth row. We repeat this *split-and-add* procedure until getting just one value. Before starting the second phase, the last value is replaced with a zero. The second phase is performed in the backward direction. We first

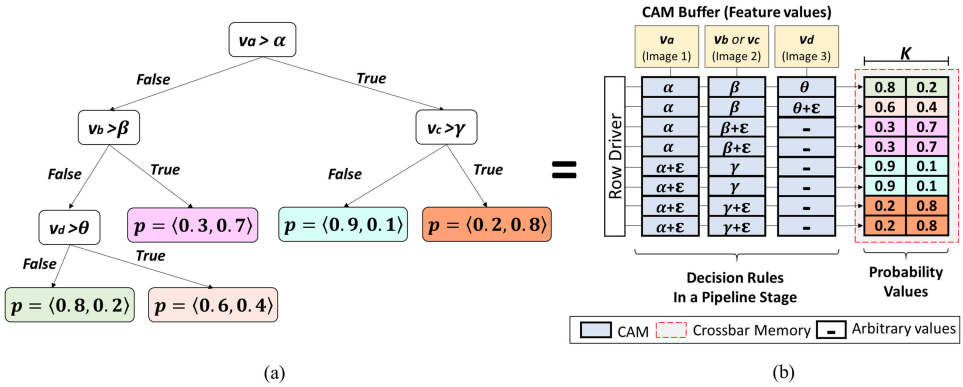

Figure 4. Example of a DT and the equivalent DT-MEM structure. (a) Decision Tree. (b) DT-MEM.

add the two rows at the end, and the last row and added row are merged into the forth row from the end. We repeat this *add-and-merge* procedure to accumulate the partial sums, and as the result, the prefix sums are stored at the first memory row. It can compute the prefix sum in $O(\log(N))$ time for any N pixels on the row or column, while utilizing the memory space in the order of $O(N\log(N))$. After computing the integral image, we can compute a Haar-like feature with two in-memory additions, i.e., $(s(A) + s(D))$ and $(s(B) + s(C))$, while the addresses are decoded from the pixel coordinates. Since the pixel sizes masked by the black and white stripes may vary, each feature is usually weighted to compensate the difference. The subsequent subtraction and weighting are processed by a small CMOS-based weighted subtractor block which implements the subtraction and weighting logic using shift operations.

In-Memory Image Classification

The ensemble algorithm examines a classification rule by combining many relatively simple learners, called weak or base learners. The sufficient number of base learners depends on applications. For example, for the face recognition, more than 2000 base learners are used.[2] The most popular choice for the weak learner is a DT. Figure 4(a) illustrates an example of a DT. This DT has multilevel decisions using three decision stumps, i.e., the three decision nodes of the tree. For each decision stump, different features of an F-dimension data point are considered, e.g., v_a, v_b, and v_c, where $0 \leq a, b, c < F$. The leaf node includes the probabilities of each class, $\boldsymbol{p}$. In this example, the number of classes K is 2.

ORCHARD design accelerates a generic DT structure used in various ensemble modeling methods, e.g., boosting, random forest, and cascade models. In the proposed design, a DT-MEM in a form of associative memory implements a DT by utilizing in-memory similarity search functionality. Figure 4(b) illustrates the DT-MEM that corresponds to the example DT. The DT-MEM has two main memory components: content addressable memory (CAM) and crossbar memory. The CAM component stores decision rule values and supports an in-memory similarity search operation[1] that finds the rows whose values are the most similar with the value of the CAM buffer. This operation is implemented by adding a column driver that activates bit lines of the memory block. The crossbar memory has the K probability values of the leaf nodes in each memory row. A column of the CAM component corresponds to decision stumps at one level of the DT. For example, the rows in the zeroth column represent the decision stump of the root node, i.e., $v_a > \alpha$. The first four rows are set by $\boldsymbol{\alpha}$, while the other four rows have $\alpha + \varepsilon$ which is a value whose all elements are 0's except the last bit of 1. The next column represents decision stumps at the second level of the DT using the half rows compared to the first column; the last-level decision stumps are stored in the last column. Each row of the crossbar memory stores the probability values corresponding to each decision rule.

DT-MEM runs by first activating all rows of the first column, while the extracted feature value of an image is fetched in the zeroth column in the CAM buffer. In the next cycle, it performs the similarity search of all the activated rows in the CAM. For example, for the root decision rule that

Published by the IEEE Computer Society

compares with α, rows whose value is smaller or equal to α are selected. Otherwise, the other rows of $\alpha + \varepsilon$ are selected. The selected rows activate the lines connected to the next column (often referred to match lines,), i.e., the comparison results are sent forwards the next column for the next level decision. Next the 1th column of CAM buffer is fetched with the feature value of the image which was processed at the 0th column. The 0th column of the buffer is replaced with the feature value of a next image to process. This procedure is iteratively repeated in a pipeline—that is, the *i*th column processes the image which was processed at the $(i-1)$th column. This pipeline design enables a DT-MEM to handle multiple images that the practical image recognition tasks often need to process, e.g., subsampled images of different regions of interest. Once each DT-MEM identifies the decision probabilities, we add the probabilities of all DT trees to identify the class with the maximum cumulated probability. We employ multiple adders in a tree structure to parallelize the addition. In our implementation, the tree-based adder can add 128 floating points numbers. When the number of DT-MEMs is larger than 128, it serially computes the sum of probabilities for each class by grouping DT-MEMs. For cascade models which have multiple groups of DTs, we execute the tree-based adder for each group and check the accumulated probabilities to decide if it needs to proceed to the next stage decision. The decision rule comparisons happen inside the memory in parallel without external accesses to the stored data for the DT model. This significantly reduces the data movement overhead, thus improving performance of the whole prediction procedure.

Table 1. Recognition models for four benchmarks.

Name	Feature Extraction	# of Features	Classifier	# of DTs
MNIST	HOG	392	AdaBoost /Random Forest	1024
Face	HOG	608		2048
Pedes-trian	Haar-like	1461	Cascade (30 Stages)	1464
Vehicle	Haar-like	250	Cascade (13 Stages)	250

EXPERIMENTAL RESULTS

Experimental Setup

We evaluate the proposed ORCHARD design by using circuit- and device-level simulations. For the circuit-level pre-RTL simulation, we use HSPICE simulator in 45-nm technology. Our design works with any bipolar resistive technology, which is the most commonly used in existing NVMs. We use VTEAM[5] for the memristor device model that has a large OFF/ON resistance ratio to provide stable and large sense margin for the in-memory operations. We perform the prelayout simulation using system verilog and synopsys design compiler in 45-nm TSMC technology. We have also cross-validated the power and performance of the crossbar memory blocks using NVSIM[6] that models fabricated NVM devices. To compare the energy and performance efficiency to existing processor-based implementations, we measured the power consumption of Intel Xeon E5440 processor and ARM Cortex A53 processor using HIOKI 3334 power meter. We exploit four image recognition datasets: text (MNIST),[7] face (Caltech 10 000 webfaces[8] and Cifar-100),[9]

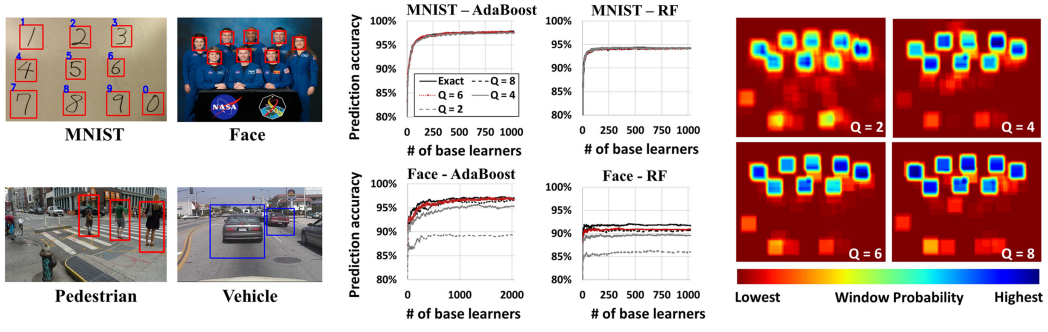

Figure 5. Image recognition quality of ORCHARD. (a) Practical Image Recognition. (b) Accuracy for different Q and L. (c) Window probability vs. approximation: See the Face image in (a).

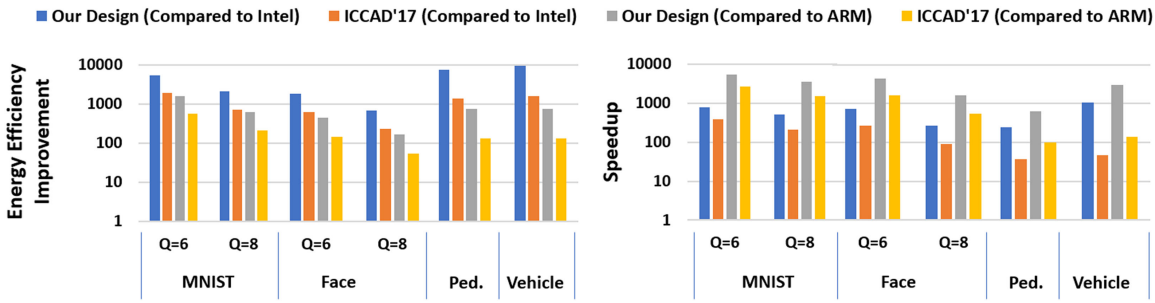

Figure 6. Energy and performance improvement.

pedestrian (INRIA),[10] and vehicle (UIUC car detection).[11] For each benchmark, we trained the model shown in Table 1 with scikit-learn and OpenCL library offline and wrote the ORCHARD memory blocks of our simulation environment. Note that the proposed design can support various DT-based ensemble models including AdaBoost, random forest, and cascade algorithms.

Image Recognition Accuracy

Figure 5(a) shows the object recognition results for four images. For the MNIST and face models, the quantization level of the HOG approximation is set to 6. As shown in the results, ORCHARD successfully recognizes the target objects. Figure 5(b) shows the prediction accuracy changes of MNIST and face when different numbers of base learners (L) and quantization levels (Q) are used. The results show that by selecting the sufficient number of base learners and quantization levels, we can achieve the same level of recognition quality as compared to the precise feature computation (denoted by Exact in the figure). For example, using AdaBoost for MNIST, there is only 0.4% error with $Q = 2$ and $L = 1024$, resulting in 97.5% accuracy. For the Face model, it can recognize images with 96.7% accuracy which incurs only 0.4% error, when $Q = 8$ and $L = 2048$. The models of the random forest (RF) are also approximated with minimal error of 1% when $Q = 8$. For the pedestrian and vehicle models, it precisely performs the recognition procedure without any approximation. For these two models, the recognition precision is 91.0% and 93.8%, respectively. To better understand the impact of the quantization levels, we compute window probability for each pixel, which is the normalized sum of the predicted probabilities for all sliding window regions. Figure 5(c) illustrates four images of the window probability for different Q values. The results show that the recognition quality increases with higher quantization level. For example, for both the $Q = 6$ and $Q = 8$ cases, we can accurately recognize faces, whereas the $Q = 2$ case is likely to create more false positives.

Energy and Performance Improvement

We evaluate energy and performance efficiency of the proposed accelerator by comparing with the existing processor-based procedure and earlier memory-based accelerator (ICCAD'17)[1] which sequentially processes each image and relies on integral images computed by the CPU. Figure 6 shows that the ORCHARD design significantly improves the efficiency of the image processing tasks. For example, when using the AdaBoost model for MNIST with $Q = 8$, it shows energy efficiency improvements of 2100× relative to the ×86 processor-based computation with 500× speedup, and 610× as compared to ARM Cortex A53 with 3500× speedup. Even for the most complex model (Face with $Q = 8$,) the proposed design can process an image within 1 microsecond. As compared to the earlier memory-based accelerator, the proposed design achieves 3× and 2.4× improvements for the energy and performance, respectively. The improvement for the cascade models is also significant, e.g., 750× energy improvement and 3000× speedup for the Vehicle workload, as compared to the ARM-based execution.

The ORCHARD accelerator can be designed with minimal area overhead to the existing NVM technology. In our evaluation, only 4% of the total area corresponds to CMOS circuitry including the microcontroller and adder/subtractors. The required memory resource is usually dependent on the size of the HOG feature extractor, which stores the precomputed data, and the

Published by the IEEE Computer Society

number of DT-MEMs. In our evaluation, the Face model with $Q = 8$ requires the largest memory size, however this model can be also implemented only 600 MB for the extractor and 67 MB for the DT-MEMs.

CONCLUSION

We propose ORCHARD, which accelerates the object recognition task by using in-memory processing. Since all main computation of the feature extraction and classification are done inside memristor blocks that consume less energy and run faster, ORCHARD can improve the efficiency significantly by energy efficiency improvement of up to 2100× and 500× speedup as compared to the CPU implementation. As compared to the previous memory-based accelerator, the proposed design presents 3× and 2.4× improvements for the energy and performance, respectively.

ACKNOWLEDGEMENT

This work was supported in part by the Center for Research in Intelligent Storage and Processing in Memory (CRISP), one of six centers in the Semiconductor Research Corporation's Joint University Microelectronics Program (JUMP), in part by an SRC program sponsored by the Defense Advanced Research Projects Agency (DARPA), and in part by the National Science Foundation (NSF) under Grant 1730158 and Grant 1527034.

REFERENCES

1. Y. Kim, M. Imani, and T. Rsing, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *IEEE Comput.-Aided Design*, 2017, pp. 25–32.
2. M. Mathias, R. Benenson, M. Pedersoli, and L. V. Gool, "Face detection without bells and whistles," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 720–735.
3. B. Bilgic, B. K. P. Horn, and I. Masaki, "Efficient integral image computation on the GPU," in *Proc. IEEE Intell. Veh. Symp. IV*, 2010, pp. 528–533.
4. M. Imani, S. Gupta, and T. Rsing, "Ultra-efficient processing in-memory for data intensive applications," in *Proc. ACM Design Automat. Conf.*, 2017, Paper 6.
5. S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 39, no. 10, pp. 786–790, Aug. 2015
6. X. Dong, C. Xu, N. Jouppi, and Y. Xie, "NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*. New York, NY, USA: Springer, 2014.
7. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
8. A. Angelova, Y. Abu-Mostafam, and P. Perona, "Pruning training sets for learning of object categories," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2005, pp. 494–501.
9. A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009, 2009.
10. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2005, pp. 886–893.
11. S. Agarwal, A. Awan, and D. Roth, "Learning to detect objects in images via a sparse, part-based representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 11, pp. 1475–1490, Nov. 2004.

Yeseong Kim is currently working toward the Ph.D. degree in computer science and engineering at University of California San Diego, La Jolla, CA, USA. His research interests include alternative computing, computer architecture, and embedded systems. He received the B.S. degree in computer science and engineering from Seoul National University, Seoul, South Korea, in 2011. Contact him at yek048@ucsd.edu.

Mohsen Imani is currently working toward the Ph.D. degree in computer science and engineering at University of California San Diego, La Jolla, CA, USA. His current research interests include approximate computing, computer architecture and embedded systems. He received the M.S. degree in electrical and computer engineering from the University of Tehran, Tehran, Iran, in 2014. Contact him at moimani@ucsd.edu.

Tajana Simunic Rosing is currently a Full Professor, a holder of the Fratamico Endowed Chair, and an IEEE Fellow. Her current research interests include energy efficient computing and embedded and wireless systems. She received the Ph.D. degree from Stanford University, Stanford, CA, USA, in 2001, concurrently with completing the M.S. degree from engineering management in 2000. Contact her at tajana@ucsd.edu