ORCHARD: Visual Object Recognition Accelerator Based on Approximate In-Memory Processing

Yeseong Kim, Mohsen Imani, Tajana Rosing
University of California San Diego
{yek048, moimani, tajana}@ucsd.edu

***Abstract*—In recent years, machine learning for visual object recognition has been applied to various domains, e.g., autonomous vehicle, heath diagnose, and home automation. However, the recognition procedures still consume a lot of processing energy and incur a high cost of data movement for memory accesses. In this paper, we propose a novel hardware accelerator design, called ORCHARD, which processes the object recognition tasks inside memory. The proposed design accelerates both the image feature extraction and boosting-based learning algorithm, which are key subtasks of the state-of-the-art image recognition approaches. We optimize the recognition procedures by leveraging approximate computing and emerging non-volatile memory (NVM) technology. The NVM-based in-memory processing allows the proposed design to mitigate the CMOS-based computation overhead, highly improving the system efficiency. In our evaluation conducted on circuit- and device-level simulations, we show that ORCHARD successfully performs practical image recognition tasks, including text, face, pedestrian, and vehicle recognition with 0.3% of accuracy loss made by computation approximation. In addition, our design significantly improves the performance and energy efficiency by up to 376x and 1896x, respectively, compared to the existing processor-based implementation.**

Keywords*—*Adaboost, object recognition, processing in-memory, non-volatile memory

I. Introduction

An important task in the emerging Internet of Things (IoT) domain is visual object recognition which analyzes images and videos to extract useful information about the surrounding environment [1]. For example, autonomous vehicles need to recognize various objects such as cars, pedestrian, and traffic signals, from real-time videos taken by cameras. Since most object recognition procedures are both data and compute intensive, general-purpose processors often do not provide enough power efficiency and performance for real-time response. An alternative solution is to execute the object recognition procedure on specialized hardware accelerators such as FPGAs. These accelerator designs rely on the CMOS-based processing logic which consumes a lot of power, and the performance is limited by the memory bandwidth. For example, a recent FPGA-based design provides only 7.5**x** speedup as compared to the processor-based implementation [2]. Considering the significant increase in visual data, e.g., more than 250 billion photos uploaded to Facebook every day [3], we need more efficient strategies to accelerate these tasks.

In this paper, we propose a novel accelerator design, called ORCHARD (Object Recognition and Classification Hardware Accelerator on Resistive Devices), which performs the object recognition computation inside memory. The proposed accelerator has computation-enabled memory blocks which store the image data and the machine learning models. We utilize the memristor NVM, which has zero leakage power consumption and fast read operations [4], to enable high power and performance efficiency of ORCHARD. The proposed design processes both the feature extraction procedure and image classification tasks, which are key subprocedures of the object recognition. We support two popular feature extraction algorithms, *HOG* (Histogram of Oriented Gradient) [5] and *Haar-like* feature extraction [6]. The feature extraction computations are modeled to memory operations and further optimized based on the idea of approximate computing. The image classification task exploits *boosting algorithm*, which is one of the best image recognition algorithms nowadays [7, 8]. The boosting algorithm belongs to a meta algorithm of *ensemble learning methods*, which utilize many simple, complementary learning models, called *base learners*. Since each base learner is independent of each other, we can highly parallelize them by modeling each learner in a memristor block.

In our evaluation, we show that the proposed ORCHARD design successfully performs four practical image recognition tasks: text, face, pedestrian, and vehicle recognition. Our experimental results also show that the proposed design significantly reduces the computation overhead. For example, with a minimal accuracy loss of 0.3% by the approximation, we increase performance and energy efficiency by up to 376**x** and 2896**x**, respectively, compared to the state-of-the-art processor-based computation.

This work was supported by NSF grants 1730158 and 1527034.

II. Related Work

Visual object recognition acceleration: To provide efficient mechanisms for the object recognition, previous work accelerates them by creating application-specific hardware. Much research focus on the acceleration of the boosting-based image classification procedure which most recent algorithms exploit [7, 8]. Several FPGA-based implementations have been proposed to enable the object detection in real-time videos, e.g., face [9], human [10, 11], cars [12], vision-based cancer diagnosis [13] and generic objects [14]. For example, [15] proposed a FPGA accelerator which perform Haar-like feature extraction and *Adaboost* algorithm for the image classification. An ASIC-based implementation has been also recently proposed to further improve performance [2]. However, the computation tasks of these designs are processed on the traditional CMOS-based, application-specific processing cores and logic, and thus still have high overhead. In addition, they perform precise computation of the recognition tasks, missing many opportunities to further accelerate. In contrast, our proposed ORCHARD processes the tasks inside memory while further leveraging approximate computing.

Another popular algorithm actively investigated nowadays for object recognition is deep neural networks. Both FPGA [16] and ASIC-based designs [17] have been proposed. Even though these neural network-based approaches show superior quality in the recognition tasks, there are still significant energy and performance issues due to the high computation complexity and large memory footprints of models. In this work, we exploit AdaBoost as an alternative solution, which

has been widely used in computer vision field. This learning method is often relatively light-weight, and shows better accuracy than DNN in some cases of the image recognition, e.g., face detection [7]. In addition, this requires less effort to tune modeling parameters and the trained models are easy to interpret, making it a viable solution for diverse object recognition tasks without losing generality.

Memory-Based Computing: *Near-data computing* can be an effective solution to accelerate computation by reducing the overhead of data movements between hardware blocks [18]. Prior research investigated Processing In Memory (PIM) as a strategy for the near-data computing. For example, the design proposed in [18] exploits a 3D-stacked structure which implements a set of instructions, e.g., floating point add and Euclidean distance, on the top of DRAM blocks. Other work showed that more sophisticated functionality can be also implemented, e.g., MapReduce based on 3D stacking [19] and nearest neighbor search using computational logic next to DRAM [20]. With advances of emerging NVM technology such as phase-change RAM (PCRAM), spin-transfer torque RAM (STT-RAM), and memristor devices, several research focused on NVM-based PIM techniques since the new memory devices show superior characteristics such as high density, low-power consumption, and scalability [4, 21]. For example, work in [22] proposed AC-DIMM which supports basic bit-serial instructions and search operations on STT-RAM. The authors in [23] presented how to support bit-wise operations in memory using analog characteristic of NVM devices. The authors in [24] modified a NVM-based content addressable memory architecture to perform in-memory brain-inspired computing by supporting nearest Hamming distance search operation. The NVM memory was also used to accelerate computation on general-purpose processors by storing precomputed results and exploiting them to reduce redundant computation [25, 26].

As compared to PIM techniques which focus on assistance of PIM-based operations for general-purpose processors, we devise a memory-based accelerator which covers a whole body of the image recognition task without using general-purpose processors. To the best of our knowledge, this is the first work which accelerates the image recognition procedure fully utilizing in-memory computing.

III. Design of ORCHARD

A. Overview of ORCHARD

Figure 1 shows an architectural overview of the proposed ORCHARD design. In an offline stage, we interpret trained models for the object recognition models, and write the model into the memory blocks of the *feature extractor* and *boost learners*. Once the model is written, ORCHARD can perform in-memory image recognition procedure at runtime for a given image, e.g., an image including a human face. The feature extractor consists of two types of crossbar memory modules, approximate HOG feature extractor and Haar-like feature extractor module, which are selectively used according to the model description, and each of which stores a set of precomputed results for each extraction. In each module, ORCHARD performs the feature extraction tasks through memory access operations by converting pixel information of the given image to memory addresses. The extracted features are written to the feature buffer of the boost learner for the classification task.

The boost learner performs the image classification task, using multiple decision tree memory blocks, in short, DT-MEM, which model the functionality of base learners of boosting algorithms. The DT-MEM block is designed as associative memory (AM), which supports in-memory search operation

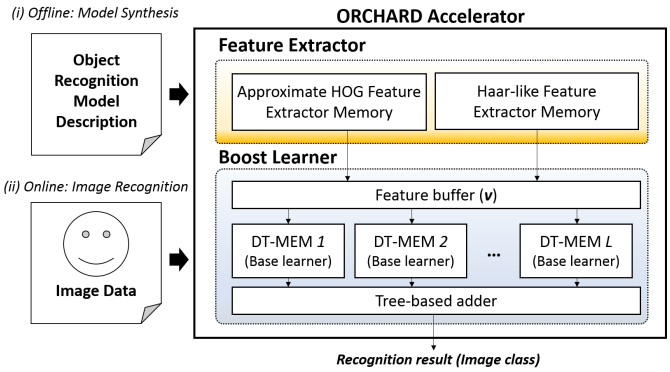


Fig. 1: An architectural overview of ORCHARD

using content addressable memory (CAM). We map each decision step of the base learner into the search operation of the DT-MEM blocks. Through the in-memory classification with the extracted features, each DT-MEM produces a list of probability values for the recognized objects as the outputs, and they are accumulated using a tree-based adder to create the final recognition result. For example, if the recognition task is to detect digital numbers among 10 classes, i.e., from 0 to 9, the corresponding 10 probability values are computed. For the two class problems, the final result includes two probabilities values, i.e., for the binary decision of 'true' and 'false' cases. Then, the identified image class which has a high probability is determined as the one including the target object.

Note that most of the subblocks in the two modules are designed using resistive memory. Since the major computation steps of the recognition procedure are implemented as memory operations, the ORCHARD can accomplish the recognition tasks by invoking the in-memory computing functionalities, while only light-weight, high-level algorithm controls happen in the microcontroller. This significantly reduces the power and performance overhead due to CMOS-based logic. In the next section, we discuss how ORCHARD processes the recognition tasks in each memory block.

B. Memory-Based Feature Extraction

In modern image recognition algorithms, a feature extraction procedure precedes the classification to identify useful information from raw image pixels, and this improves the quality of the recognition. The proposed ORCHARD implements two most popular feature extraction procedures, HOG and Haar-like features, in the memory blocks.

1) Approximate HOG Feature Extraction: Before explaining the proposed PIM technique for the HOG extraction, we describe the basics of the original HOG extraction algorithm. The underlying intuition is that the distribution of pixel gradients in a small region can describe shape and appearance of target objects. Figure 2 illustrates the HOG feature extraction procedure with an example. The procedure first divides the original image into multiple *regions*. The 32x32 image in the figure is divided into four regions. For each region, it computes the gradient values of all pixels by considering its adjacent pixels, called *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

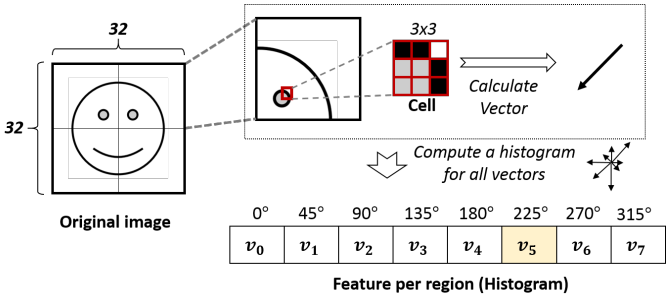
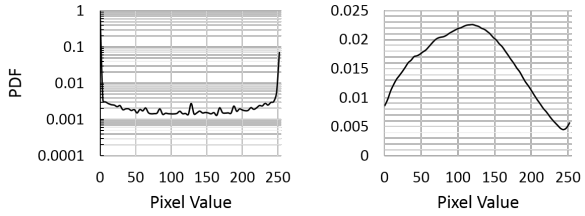


Fig. 2: An illustration of HOG feature extraction



(a) MNIST (b) WebFaces

Fig. 3: Pixel distribution of two benchmarks

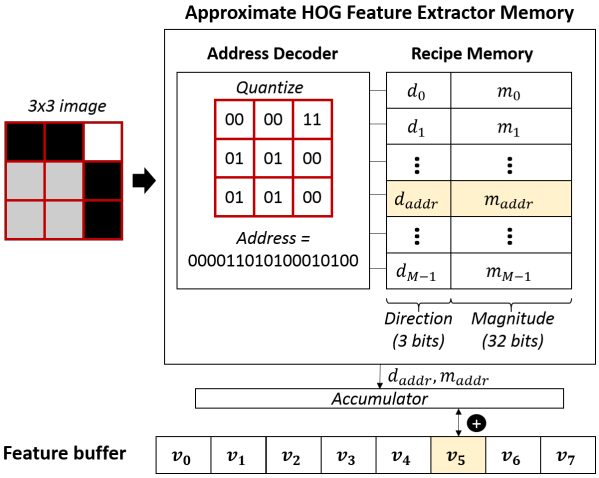


Fig. 4: Approximate in-memory HOG extraction

number of regions and B is the number of bins. If the image includes multiple color channels, we may consider either the grayscale image or each color channel separately. In this procedure, the main bottleneck is the vector computation, as it typically involves many arithmetic operations, e.g., gradient calculation and trigonometrical functions. In our ORCHARD design, we optimize the vector computation by modeling and approximating it as a single memory access.

Let f be a function to be modeled, where it takes an operand in a set $\mathbb{X}$ and produces an output in a set $\mathbb{Y}$, i.e., $f : \mathbb{X} \rightarrow \mathbb{Y}$. If the input set $\mathbb{X}$ is finite, the output set $\mathbb{Y}$ is finite. Thus, we may imagine a memory which stores all the precomputed results of $\mathbb{Y}$ while the memory address is given by indexing each element of $\mathbb{X}$. In our case that the gradient vector computation is the function f, 256^9 memory rows could represent the function, since there are 256 values for a pixel of an image color channel and the cell includes 9 pixels. However, this memory size for all pixel combinations is prohibitively huge in practice.

We optimize this memory size by storing only approximate and representative values. For example, if the goal of the image recognition task is to detect a hand-written alphabet, we would approximate the input pixels using two values, e.g., for black and white, respectively. In that case, the number of required memory rows is only $2^9 (= 512)$. To verify this idea on practical images, we conducted an analysis for two popular image datasets, MNIST for text recognition [27] and Caltech 10000 web faces (WebFaces) [28]. Figure 3 summarizes the pixel distribution as a probability density function (PDF) for each dataset. As shown in the results, many similar pixels exist for the images. For example, more than 87% of pixels of the MNIST images are either 0 and 255. For the WebFaces dataset, many pixels have similar values in the middle range. This similarity allows to approximately model the complex function computation using a reasonably small memory size.

Figure 4 shows the memory structure which implements PIM-based approximate HOG feature extraction procedure. For a given 3x3 cell image, the address decoder quantizes each pixel p into a value q of a few bits, where $q = \lfloor p/(256/Q) \rfloor$ and Q is the quantization level. In this example, when $Q = 4$, the 256 pixel values are quantized to 4 values, 00, 01, 10, and 11. The quantized value are concatenated to form a memory address which 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. They are precomputed using the median values of each quantized range. Intuitively, the precomputed results keep similarity to the one precisely computed for original cell pixels. The selected row is accessed by a small CMOS-based accumulator logic block which computes the histogram of the region. This logic accumulates the magnitude m_i to a histogram bin selected by the bin index d_i, i.e., v_{d_i}. The microcontroller of ORCHARD applies each cell of the image to this memory block in a pipelined manner. Then the oriented histograms are computed into the feature buffer.

Note that all the memory rows of Q^9 for the 3x3 cell can be completely precomputed offline regardless of datasets. We design the recipe memory with memristor devices which exhibit low power and fast read access times that are comparable to SRAM. In Section III-B3, we discuss how to further optimize the crossbar memory based on the access characteristics.

2) In-Memory Haar-like Feature Extraction: ORCHARD also supports Haar-like feature extraction procedure. Figure 5 describes the original procedure and its memory-based implementation. 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 are different from the facial color. For one image, multiple features are extracted by computing a weighted difference between the pixel sum of the black stripes and that of the white stripes, where the weights for each pixel sum are used for compensating the differences in the stripe size. Since computing a sum of pixels in a stripe directly from the original pixels is cost-ineffective, the state-of-art algorithm utilizes an *integral image* [6]. The integral image has the same size to the original images, and can be calculated in $O(n)$ time where n is the number of pixels. Let assume that each pixel of the original image is $p_{(x,y)}$. Each value of the integral image $s_{(x,y)}$ is computed by $s_{(x,y)} = p_{(x,y)} + s_{(x-1,y)} + s_{(x,y-1)} + s_{(x-1,y-1)}$ where $s_{0,0}$ is assumed to be zero. Then, 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

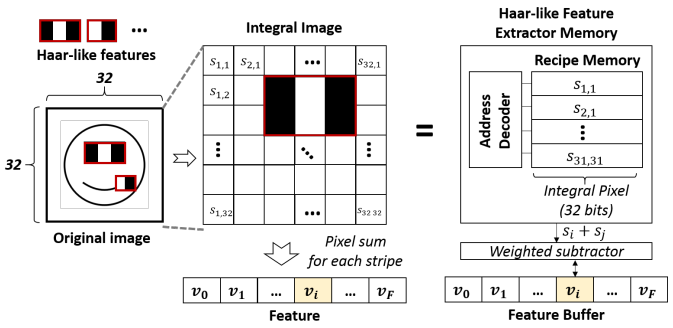


Fig. 5: Memory-based computation of Haar-like feature extraction

of a stripe can be calculated by $(s_A + s_D) - (s_B + s_C)$, where A, B, C and D are the top-left, top-right, bottom-left, and bottom-right coordinate, respectively. The procedure computes the pixel sums of the other stripes as well to determine a feature value.

ORCHARD exploits the integral image to compute the Haar-like features. As shown in Figure 5, the Haar-like feature extractor memory has a similar structure to the HOG extractor, and the crossbar recipe memory stores the integral image computed from the microcontroller. In our design, we optimize the pixel sum computation by utilizing a PIM-based addition design presented in [29]. This design performs the addition operation of two activated rows inside memory in a single cycle without using CMOS logics. Thus, we can compute a Haar-like feature from the two in-memory additions, i.e., $(s_A + s_D)$ and $(s_B + s_C)$, while the addresses are decoded from the pixel coordinates. 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.

Unlike the HOG extractor, this memory block has to be initialized with the integral image, creating additional write operations. Since the two feature extractor memories have these different characteristics, we design each memory with different optimization goals. The next section describes how the crossbar memory blocks are designed by the strategy.

3) Crossbar Memory Optimization: The crossbar memory blocks used in the proposed in-memory HOG and Haar-like feature extractors have different memory access characteristics. Thus, on the top of the benefit came from the memory-based computing, there is a potential room for efficiency improvement by further optimization. For example, the HOG extractor block can be optimized for the read latency by configuring design knobs, e.g., buffer, sense amplifier design, etc. [30], since the recipe memory can be stored once offline. In contrast, the Haar-like extractor block is optimized for the write latency to efficiently serve the runtime initialization of the integral image. Our evaluation shows that these optimizations can further improve the characteristics of these two memories. For example, for the HOG extractor memory block with 1 MByte memory, the energy and delay of a read operation are improved by 1.8× and 2.9× respectively. Similarly, we can optimize the energy and latency of write operations of the Haar-like feature memory by 4.6× and 2.4×.

C. Memory-Based Boosting Decision

The boosting algorithm generates a classification rule by combining many relatively simple learners, called *weak* or *base learners*. Each base learner is trained to cover less-accurate hypothesis of other base learners, and creates K probability values for each class, where K is the number of classes. The final *strong* prediction is made by adding the probability values of all base learners, and usually outperforms a single learner-based approach. The sufficient number of base learners, L, depends on applications. For example, for the face recognition, more than 2000 base learners are used [7]. The base learner can be any learning algorithm. Since the most popular choice for the object recognition problems is a decision tree (DT), we design a memory block capable of the decision tree processing, called DT-MEM, as the key building block of the boosting algorithm. In the following subsections, we first explain the classification procedure with the DT-MEM structure (Section III-C1), and then describe a new CAM hardware design, called *SVCAM*, which identifies the most similar values for multiple memory rows (Section III-C2).

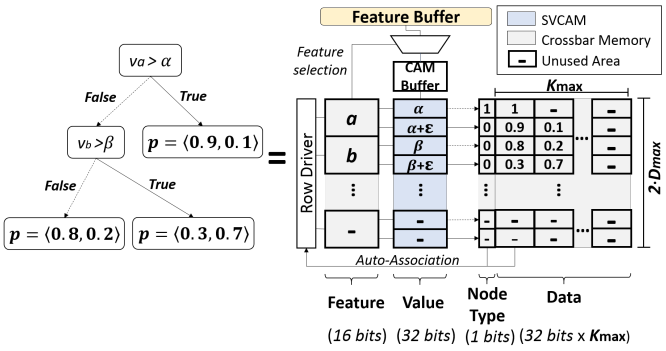
1) Boost Learner Using In-Memory Decision Tree: Figure 6a illustrates an example of a decision tree. This decision tree has multi-level decisions using two *decision stumps*, i.e., the two decision nodes of the tree. For each decision stump, different features of a F-dimension data point, $\mathbf{v}$, are considered, i.e., v_a and v_b for $0 \leq a < F$ and $0 \leq b < F$. The leaf node includes the probabilities of each class, $\mathbf{p}$. In this example, K is 2.

In the ORCHARD design, a DT-MEM implements a decision tree based on the concept of *auto-associative memory* which repeatedly activates a row using internal memory data. Figure 6b shows the structure of the DT-MEM that does the prediction of the example decision tree inside the memory. The DT-MEM has four memory components, feature, value, node type, and data. The value memory component exploits the SVCAM structure, which supports an in-memory search functionality, and others are designed as normal memory blocks. Each component except the feature one has $2D_{max}$ memory rows for each edge, while the feature part has D_{max} rows, a design parameter that determines the maximum number of decision stumps. There is another parameter, K_{max}, which is the maximum number of classes supported by ORCHARD.

Each row of the feature part corresponds to one decision stump, and its connected two rows of other parts include information about the two children nodes of the decision stump. For the decision tree shown in Figure 6a, to represent the decision stump of the root node, i.e., $v_a > \alpha$, the 0-th row of the feature part stores the feature index (32-bit integer), i.e., a, and the connected two rows of the value part are set by α and $\alpha + \epsilon$ (32-bit floating point values), respectively. ϵ is a 32-bit number whose all elements are 0's except the last bit of 1. The 1-th row of the value part and its connected row pair have the information about the decision stump of $v_b > \beta$ in the same way.

The data part stores either i) a row index (32-bit integer) if the child is another decision stump or ii) probability values $\mathbf{p}$ (32-bit floating point array) if the child is a leaf node. The child type, i.e., decision stump or leaf, is indicated by the 1-bit flag stored in the node type part. For example, since the left child of the root node is another decision stump, the node type flag is set to 1, and the first of 32 bits of the *data* part store the row index of the child decision stump, i.e., 1. In contrast, for the right child, which is a leaf node with the probability values, the node type flag is set by 0 and the data part stores $\mathbf{p}$.

Based on this structure, the DT-MEM performs the in-memory decision task by following iterative steps. (i) It first starts by activating one row. During the initial run, the first row is activated for the decision stump of the root node. Then, a feature v_t is selected based on the index t of the feature part, and placed into the CAM buffer. At the same time, the two



(a) Decision Tree (b) DT-MEM

Fig. 6: A decision tree example and equivalent DT-MEM structure

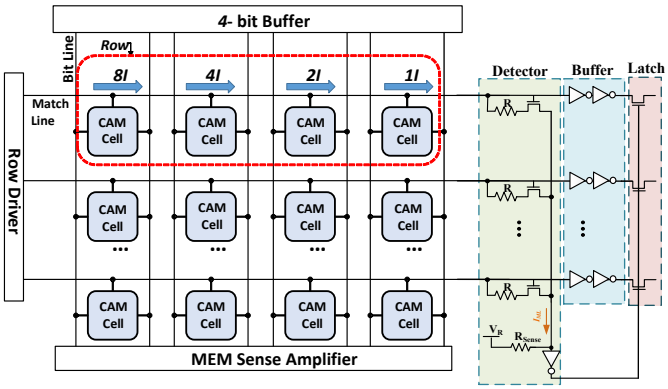


Fig. 7: 4-bit SVCAM structure

rows connected to the value part are enabled. (ii) In the next cycle, the value part performs the SVCAM-based similarity search for the two enabled rows, e.g., α and $\alpha + \epsilon$; that is, the first row (dotted arrow) is selected if the value is smaller or equal than α. Otherwise, the second row (solid-line arrow) is selected. Thus, for the node type and data part, only one row is activated. (iii) Once the one row is activated, its node type bit determines whether it proceeds the further search. In this example, if $v_a \leq \alpha$, the activated flag is 1, meaning that it needs to test another decision stump. The row index of the next decision stump is stored in the first 32-bits of the *data* component. The row driver decodes the row index and process the decision stump by going back to the step (i).

By processing the iterations until the `node type` flag is 0, we can identify probability values as the result of the DT-MEM. All these computations happen inside the memory without external accesses to the stored data for the decision tree model. This reduces the data movement overhead, thus significantly improving performance of the whole prediction procedure which many base learners involve.

Tree-based Adder Once each DT-MEM identifies the decision probabilities in parallel, `ORCHARD` adds the probabilities of all DT trees to create the final prediction as discussed in Section III-A. 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.

Cascade Decision Some object recognition models may use *Cascading* classifier algorithm, a variant of boosting algorithm, mostly with the Haar-like features. The cascading classifier also includes many base learners, but the decision is made through *multiple stages* in which a group of base learners is involved. For example, where there are L base learners, i.e., DT-MEM$^1, \cdots,$ DT-MEM^L in our design, an i-th stage has l_i base learners and $\sum l_i = L$. In the cascading procedure, the decision trees of the first stage perform the initial classification. Then, by comparing the classification result with a trained threshold, it decides whether it proceeds to the next stages or not. This multi-stage decision is repeated until the last stage. `ORCHARD` design also supports the cascading decision. After executing all the DT-MEMs of the model, we activate the tree-based adder for the results of the first stage, DT-MEM^1. This procedure is repeated in the same way as in the original cascade classification model to get the final prediction.

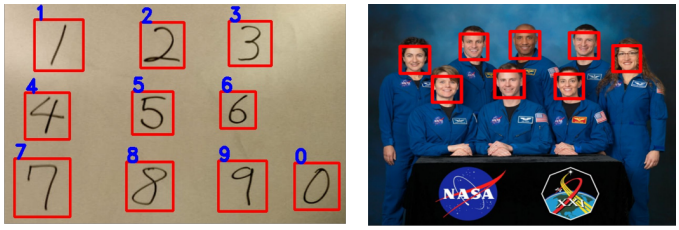
2) SVCAM design: As explained in the previous section, the SVCAM performs the comparison operation of the DT-MEM based on similarity search. Figure 7 shows an illustration of the 4-bit SVCAM as an example. The CAM cells are designed using the resistive memory, and connected each other to two types of lines, horizontal match lines (ML) and vertical bit lines (BL). A single cell stores one bit, and a row of 4 cells represents one 4-bit numeric value, which is accessible via BL using the MEM sense amplifier. The CAM functionality is performed with three logic components: input buffer, row driver, CAM sense amplifier. The input buffer stores an 4-bit value to be searched. The row driver selects two rows which are compared with the reference value. The CAM sense amplifier is used to recognize the row which has the closer value.

The CAM sense amplifier exploits analog characteristics of memristor devices to enable the search capability. The search operation starts with precharging all MLs to V_{dd}, while the buffer strengthens the bit signals of the reference value to make sure that all CAM rows can receive them at the same time. Then, each CAM cell, whose stored bit is mismatched to the reference value bit, discharges current of the corresponding ML. The discharging speed of an ML depends on the amount of current that the mismatches create. For example, if all CAM cells create the same current when mismatched, the row with the minimum bit difference discharges last. By keeping track of all MLs, we may find the row that discharged last, but this requires a complicated sense amplifier circuit such as high precision detectors and counters. To design a light-weight sense circuitry, we exploit an alternative CAM design proposed in [31], which stores the values in the inverse mode. Using this CAM design, when the input data is matched with the stored CAM value, it discharges the ML, while the ML stays charging when there is a mismatch. This simplifies our sense circuit design. We can find the nearest row by identifying the row that discharged first. Figure 7 also shows our sense amplifier design. The design consists of three main blocks: (i) a detector circuit which samples the voltage of all MLs to find the first discharged row, (ii) a buffer which delays the ML voltage to the output node, and (iii) a latch block which keeps the buffer output when the detector circuit is signaled.

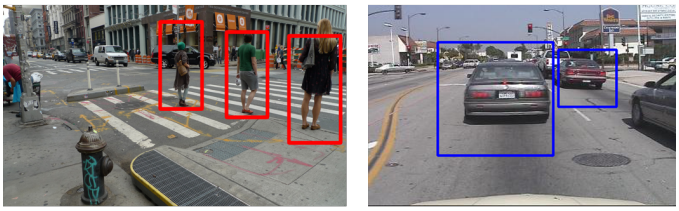
With the sense circuitry, the SVCAM design implements the similarity search function by considering the impact of each bit index on the value comparison. In our design, each cell has different memristor sizes according to its bit index, so that they create different amount of discharging current, i.e., $8I$, $4I$, $2I$, and $1I$ as denoted in the figure. For example, voltage of the $15I$ case, i.e., all 4-bit matches, drops much more quickly than in the $1I$ case. To support 32 bit floating-point number, we exploit two of the 4-bit SVCAM blocks for the exponent and six for the mantissa by connecting the latch and the row driver of adjacent blocks. Then, by activating each block serially, we find the final row which has the closest value to the reference.

TABLE I: Experimented recognition models (DS: decision stump)

Name	Feature	# of features	Classifier	# of DTs
MNIST	HOG	392	Boost with 6-level DTs	1024
Face	HOG	608	Boost with 6-level DTs	2048
Pedstrain	Haar	1464	Cascade (30 stages, DS)	1464
Vechile	Haar	250	Cascade (13 stages, DS)	250



(a) MNIST (b) Face



(c) Pedestrian (d) Vehicle

Fig. 8: Image recognition quality for four models

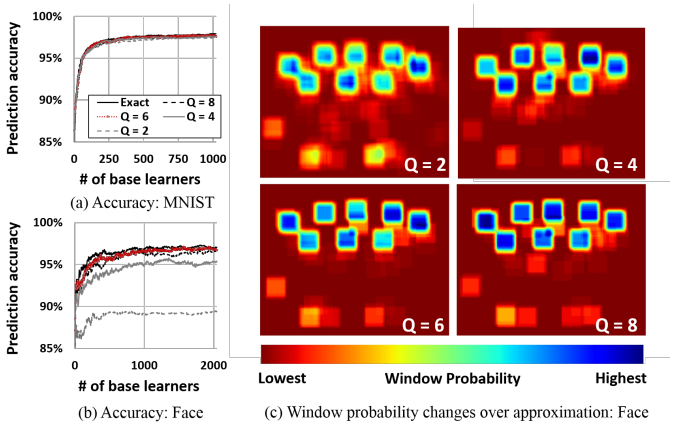
IV. Experimental Results

A. Experimental Setup

We evaluate the proposed ORCHARD design by using circuit- and device-level simulations. For the circuit-level simulation, we use HSPICE simulator in 45 nm technology. The memristor devices are designed with a large OFF/ON resistance ratio to provide stable and large sense margin [32]. The microcontoller is designed using System Verilog and Synopsys Design compiler in 45nm TSMC technology. All designed circuits have been verified by considering 10% process variations on the transistors with 5000 Monte Carlo simulations. We have also cross-validated the computed energy consumption and performance of the crossbar memory blocks using NVSIM [30]. Since the circuit-level simulation does not produce the recognition results in a reasonable execution time for practical datasets which include a large number of images, we also implemented a cycle-accurate device-level simulator, which performs the functionalities of the designed memory blocks in the software, using C++. To compare the energy and performance efficiency to existing processor-based implementations, we also measured the power consumption of Intel Xeon E5440 processor of Intel SR1560SF server and ARM Cortex A53 processor of Raspberry Pi3 using HIOKI 3334 power meter for each instrumented platform running software recognition procedures.

To verify recognition quality of ORCHARD for practical image recognition problems, we consider four problems: text, face, pedestrian, and vehicle recognition. Table I summarizes the recognition models for each recognition problem.

MNIST MNIST [27] is a canonical dataset which includes hand-written digit images. To train the model with the proposed approximate HOG features, we modify scikit-learn machine learning library [33] and create models with the training



(a) Accuracy: MNIST

(b) Accuracy: Face (c) Window probability changes over approximation: Face

Fig. 9: Accuracy changes for different HOG approximation levels

dataset using SAMME.R Adaboost algorithm [34] and 6-level decision trees as base learners. The HOG features are extracted by dividing original 28x28 pixels into 7x7 regions.

Face We exploit Caltech 10000 web faces dataset [28], and train the model in a similar way to MNIST. Negative training images, i.e., non-face images, are selected from Cifar-100 [35] and Pascal VOS 2012 datasets [36]. We select 10% of images for the testing dataset which are completely separated from the training dataset. For the HOG feature extraction, we divide a 32x32 image to i) 2x2 regions for three color channels and ii) 8x8 regions for grayscale.

Pedestrian We exploit the human body recognition model available in OpenCV library [37]. The model is built based on the Haar-like features and cascade classifier. To verify the recognition quality, we use INRIA image datasets [5].

Vehicle We exploit an OpenCV vehicle recognition model published in [38], which uses Haar-like features and cascade classifier. The recognition quality is verified with UIUC car detection dataset [39].

The models specified with either sklearn or OpenCV XML files are interpreted and stored into the memory blocks of the ORCHARD simulation environment.

B. Object Recognition Accuracy

We first present how ORCHARD recognizes visual objects in practical images. Figure 8 shows the object recognition results for four images. We loaded each model into the ORCHARD device-level simulator, and applied each image using the sliding window approach which generates multiple tested regions across an image. For the MNIST and Face models, the quantization level Q of the HOG approximation is set to 6. As shown in the results, ORCHARD successfully recognizes the target objects. For example, using the Face model which exploits the approximate feature extraction procedure, we can detect all faces.

We have also quantified detection quality of each model with its testing dataset described in IV-A. The testing dataset includes images which are not used for the model training. Figure 9a and 9b show the prediction accuracy changes of MNIST and Face respectively, for different numbers of base learners (L) and quantization levels (Q), where the accuracy is defined by the percentage of images whose object classes are correctly predicted. 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 *Exact*

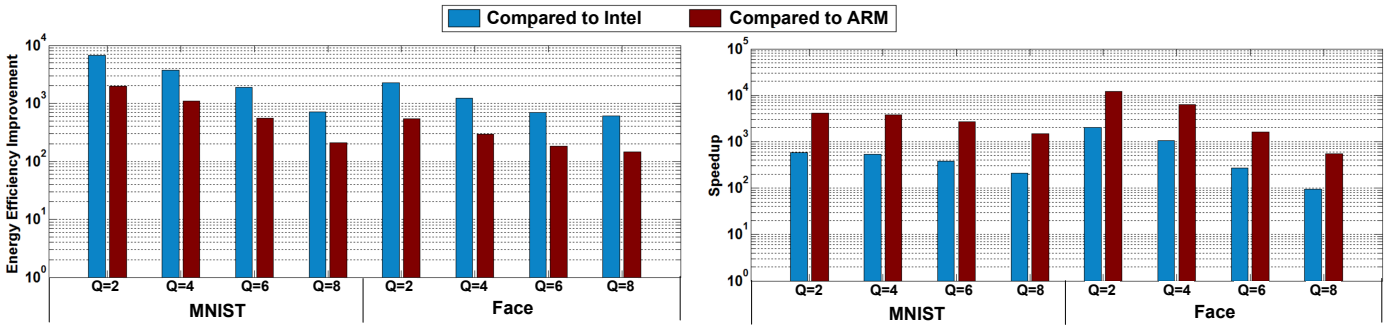


Fig. 10: Energy and performance improvement of MNIST and Face

in the figure). For example, for MNIST, there is only 0.4% accuracy loss with the most aggressive quantization level, i.e., $Q = 2$ and $L = 1024$, resulting in 97.5% of accuracy. Face model is more sensitive to the quantization level, but ORCHARD can recognize images 96.7% of the time which incurs only 0.3% error when $Q = 6$ and $L = 2048$. For the Haar-like feature-based cascade models, i.e., Pedestrian and Vehicle, ORCHARD 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 level on the recognition quality, we compute *window probability* for each pixel, which is the normalized sum of the predicted probabilities for all sliding windows that include the pixel. Figure 9c illustrates four images of the window probability for different Q values, using the same image shown in Figure 8b. The results show that the recognition quality increases with higher quantization level. For example, for both the $Q = 6$ and $Q = 8$ cases, ORCHARD can accurately recognize faces, whereas the $Q = 2$ case is likely to create more false positives. Quantization level affects area overhead of ORCHARD design and its efficiency. For example, the required size of the crossbar memory used for HOG extractor is around 45 MBytes and 600 MBytes for $Q = 6$ and $Q = 8$ cases, respectively. We present more details about ORCHARD's system efficiency for different quantization levels in the next section.

C. Energy and Performance Improvement

We evaluate energy and performance efficiency of ORCHARD accelerator by comparing with the existing processor-based object recognition procedure. We consider two processors Intel Xeon E5440 and ARM Cortex A53, which could be used in a cloud server and an embedded device platform, respectively. Figure 10 shows the energy and performance improvement of MNIST and Face running on the proposed ORCHARD accelerator as compared to the two processor-based implementations. For the two models which use the approximate HOG feature extractor, the quantization level affects the power and performance efficiency of ORCHARD since it uses less memory area. When using $Q = 6$, which has 0.3% of accuracy loss for MNIST workload, ORCHARD achieves energy efficiency improvements of 1896**x** relative to the server with 376**x** speedup, and 552**x** as compared to ARM Cortex A53 with 2654**x** speedup. The energy and latency of ORCHARD for this workload are 29 μJ and 2.0 μs. These results show that the proposed ORCHARD which executes all the tasks inside memory is very effective in terms of the processing efficiency for the object recognition.

The comparison results of the Haar-like feature-based cascade models, used for Pedestrian and Vehicle workloads, are summarized in Figure 11. These cascade models have lower efficiency improvement than the acceleration of the

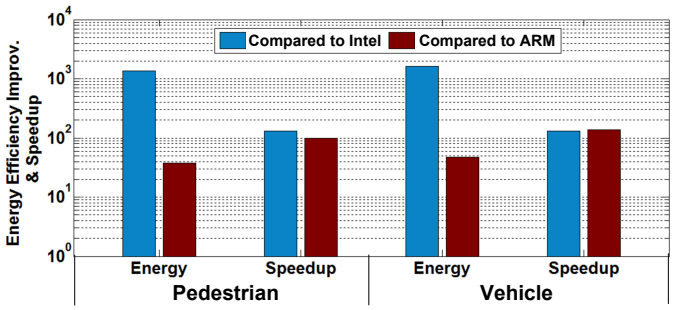


Fig. 11: Energy and performance improvements of Pedestrian and Vehicle

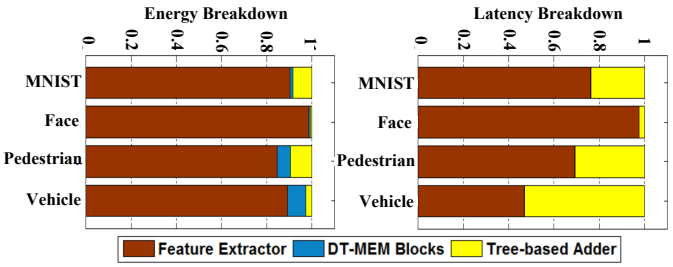


Fig. 12: Energy and latency breakdown

HOG-based boosting models. There are three main reasons for this: (i) the cascade algorithm terminates the classification procedure when inaccurate prediction results are observed in a certain stage, thus finishing faster, (ii) the decision tree has only one-level node, and (iii) ORCHARD accelerator has to compute and write the integral image to its extractor memory. However, even given this, the improvements relative to processor-based implementations on server (embedded device) are significant, e.g., 1352**x** (132**x**) energy improvement and 37**x** (98**x**) speedup for the Pedestrian workload.

To understand the energy and performance efficiency of ORCHARD accelerator, we further separated the energy and latency of the prediction procedure into three components: the feature extractor, DT-MEM blocks, and tree-based adder in the boost learner. Figure 12 shows the breakdown of energy and latency for the four models when $Q = 6$. The results show that a relatively large portion of energy and execution time is consumed by the crossbar memory in the feature extractor, since it requires many memory operations. Haar-like feature extractor consumes 93% power to write the integral image during the initialization stage. In contrast, the latency of write operations take less portion (63% of time) than energy in this memory block, since we optimize the crossbar memory for the write latency. This presents that the memory block

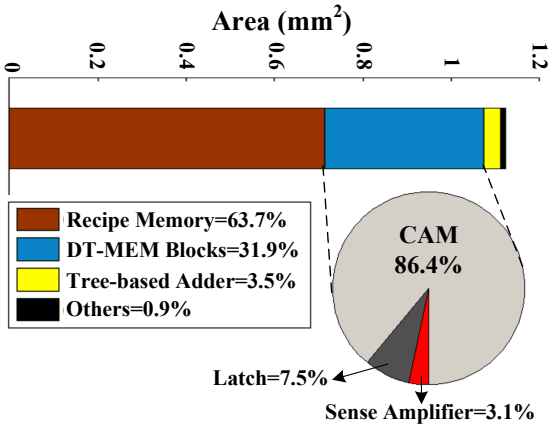


Fig. 13: Area overhead analysis

optimization discussed in III-B3 also contributes the overall improvement of the performance efficiency. Once the features are extracted, DT-MEMs can perform classification in parallel, taking on average only 4% of the energy and 0.2% of the total latency for the four recognition models.

D. Area Overhead

We also estimate the area overhead of ORCHARD. Here, ORCHARD is designed so that it can support all four models with $Q = 6$. Figure 13 shows that the crossbar memory blocks of the two feature extractors and all the DT-MEM blocks take 63.7% and 31.9% of the total area, respectively. The rest, totaling 4.4% area, corresponds to other CMOS-based circuitry, including the tree-based adder, microcontroller, adder/subtractor of the feature extractor, and MUX. This analysis shows that the proposed ORCHARD can be designed with minimal CMOS logic which typically consumes a lot of energy in the existing processors.

V. Conclusion and Discussion

In this paper, we propose ORCHARD, which accelerates the object recognition task by using approximate in-memory processing. The proposed design accelerates the two key procedures of the recognition task: the feature extraction and boosting-based classification algorithm. We use approximation techniques for HOG feature extraction and optimize for the different memory access characteristics of the two feature extractor modules. Since all main computation of the classification and recognition tasks is done inside memristor blocks that consume less energy and run faster, we can improve the efficiency significantly. In our evaluation conducted with four practical image recognition tasks, we show that ORCHARD accelerator accurately detects the target object, while achieving energy efficiency improvement of up to 1896**x** and 376**x** speedup as compared to the server-based implementation with only 0.3% accuracy loss. Our current ORCHARD design can be generalized as a solution for diverse classification problems, which also exploits ensemble of decision trees, since it can accelerate any decision tree. In addition, the in-memory similarity search could be used as a basic operation to further accelerate of the model training procedure as well.

VI. References

[1] James Manyika, et al. Big data: The next frontier for innovation, competition, and productivity. 2011.

[2] Shanlin Xiao, et al. Design of an efficient asip-based processor for object detection using adaboost algorithm. In *Information and Communication Technology for Embedded Systems (IC-ICTES), 2016 7th International Conference of*. IEEE, 2016.

[3] Bart Thomee, et al. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 2016.

[4] Yeseong Kim, et al. Cause: critical application usage-aware memory system using non-volatile memory for mobile devices. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 690–696. IEEE Press, 2015.

[5] Navneet Dalal et al. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. IEEE, 2005.

[6] Paul Viola et al. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[7] Markus Mathias, et al. Face detection without bells and whistles. In *European Conference on Computer Vision*. Springer, 2014.

[8] Shanshan Zhang, et al. Filtered channel features for pedestrian detection. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015.

[9] Shouyi Yin, et al. An adaboost-based face detection system using parallel configurable architecture with optimized computation. *IEEE Systems Journal*, 2015.

[10] Kazuhiro Negi, et al. Deep pipelined one-chip fpga implementation of a real-time image-based human detection algorithm. In *Field-Programmable Technology (FPT), 2011 International Conference on*. IEEE, 2011.

[11] DOHI Keisuke, et al. Fpga implementation of human detection by hog features with adaboost. *IEICE TRANSACTIONS on Information and Systems*, 2013.

[12] Filip Kadlček et al. Fast and energy efficient adaboost classifier. In *Proceedings of the 10th FPGAworld Conference*. ACM, 2013.

[13] Juan Silva, et al. Toward embedded detection of polyps in wce images for early diagnosis of colorectal cancer. *International Journal of Computer Assisted Radiology and Surgery*, 2014.

[14] Yao-Tsung Yang et al. Boosted multi-class object detection with parallel hardware implementation for real-time applications. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014.

[15] Christos Kyrkou et al. A flexible parallel hardware architecture for adaboost-based real-time object detection. *IEEE Transactions on very large scale integration (VLSI) systems*, 2011.

[16] Suyog Gupta, et al. Deep learning with limited numerical precision. In *ICML*, 2015.

[17] Tao Luo, et al. Dadiannao: A neural network supercomputer. *IEEE Transactions on Computers*, 2017.

[18] Junwhan Ahn, et al. Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*. IEEE, 2015.

[19] Seth H Pugsley, et al. Ndc: Analyzing the impact of 3d-stacked memory+ logic devices on mapreduce workloads. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 2014.

[20] Carlo C del Mundo, et al. Ncam: Near-data processing for nearest neighbor search. In *Proceedings of the 2015 International Symposium on Memory Systems*. ACM, 2015.

[21] Seyedhamidreza Motaman, et al. A novel slope detection technique for robust sttram sensing. In *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pages 7–12. IEEE, 2015.

[22] Qing Guo, et al. Ac-dimm: associative computing with stt-mram. In *ACM SIGARCH Computer Architecture News*. ACM, 2013.

[23] Mohsen Imani, et al. Mpim: Multi-purpose in-memory processing using configurable resistive memory. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pages 757–763. IEEE, 2017.

[24] Mohsen Imani, et al. Exploring hyperdimensional associative memory. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pages 445–456. IEEE, 2017.

[25] Mohsen Imani, et al. Approximate computing using multiple-access single-charge associative memory. *IEEE Transactions on Emerging Topics in Computing*, 2016.

[26] Vahideh Akhlaghi et al. Resistive bloom filters: from approximate membership to approximate computing with bounded errors. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 1441–1444. EDA Consortium, 2016.

[27] Yann LeCun, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[28] Anelia Angelova, et al. Pruning training sets for learning of object categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. IEEE, 2005.

[29] Mohsen Imani, et al. Ultra-efficient processing in-memory for data intensive applications. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 6. ACM, 2017.

[30] Xiangyu Dong, et al. Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory. In *Emerging Memory Technologies*. Springer, 2014.

[31] Mohsen Imani, et al. Resistive cam acceleration for tunable approximate computing. *IEEE Transactions on Emerging Topics in Computing*, 2016.

[32] Shahar Kvatinsky, et al. Vteam: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2015.

[33] Scikit-learn machine learning in python. http://scikit-learn.org/stable/.

[34] Ji Zhu, et al. Multi-class adaboost. *Statistics and its Interface*, 2009.

[35] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[36] M. Everingham, et al. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[37] Opencv. http://opencv.org/.

[38] M Oliveira et al. Automatic detection of cars in real roads using haar-like features. *Department of Mechanical Engineering, University of Aveiro*, 2008.

[39] Shivani Agarwal, et al. Learning to detect objects in images via a sparse, part-based representation. *IEEE transactions on pattern analysis and machine intelligence*, 2004.