Energy-Efficient Pedestrian Detection System: Exploiting Statistical Error Compensation for Lossy Memory Data Compression

Yinqi Tang, *Student Member, IEEE*, and Naveen Verma, *Member, IEEE*

***Abstract*— Pedestrian detection represents an important application for embedded vision systems. Focusing on the most energy constrained implementations, systems have typically employed histogram of oriented gradients features and support vector machine classification, which leads to low detection accuracy (a log-average miss rate of 68% on the Caltech Pedestrian dataset). Additionally, single-scale detection is often adopted in these systems for real-time processing, which further deteriorates the detection performance. In this paper, we propose a hardware accelerator achieving substantially higher detection accuracy by employing aggregated channel features (ACFs) at multiple different scales and using boosted decision trees for classification. Though resulting in higher accuracy, the higher dimensionality ACFs exacerbate memory operations, which become the energy and speed bottlenecks of the system. To overcome this, we employ binary discrete cosine transform to perform low-overhead and lossy compression, to efficiently store and access feature data. For restoring performance following compression, we exploit retraining of the classifier, resulting in an optimal model for pedestrian detection. The proposed accelerator is implemented in field-programmable gate array, which can process 40 video graphics array frames (640 × 480 resolution) per second at a log-average miss rate of 42% on the Caltech Pedestrian dataset, with compression reducing memory energy by 4× and overall energy by 1.7×.**

***Index Terms*— Energy efficiency, error compensation, field-programmable gate array (FPGA), lossy compression, pedestrian detection.**

I. Introduction

PEDESTRIAN (human) detection from images or videos is necessary for various embedded applications, such as advanced driver assistance systems [1], surveillance, portable electronics, robotics, and the Internet of Things [2]. These applications often require the detectors to be accurate, real-time at high frame rates, and energy efficient. This has spurred the development of hardware accelerators for detection. Among these requirements, low-energy consumption is becoming increasingly important because of the heat and battery limitations for embedded computing systems [3]. However, high detection accuracy typically requires extracting high-dimensional feature data from images or videos, which need large embedded or, in some cases, off-chip memories for storage. This poses an energy bottleneck in current integrated circuit (IC) technologies, due to the high cost of memory operations (even with modest-sized arrays) compared with computational operations [4]. In addition, these large feature data often result in increased number of memory accesses, thus limiting the throughput of the system.

An important opportunity that is being recognized in machine learning algorithms is statistical error tolerance. On the one hand, this can arise implicitly in machine learning systems, due to the possibly large margin between different decision values. In particular, high-dimensional data typically exhibit redundancy in their representations [5]. On the other hand, error tolerance can also be actuated explicitly, for instance, through data-driven training of a machine learning model to specifically compensate for the errors [6].

The objective of this paper is to evaluate an architectural approach, whereby this opportunity is specifically directed toward addressing the bottleneck of memory operations in embedded vision systems, which could show great leverage by helping to address the energy and performance constraints in various emerging applications. The proposed architecture exploits aggressive compression on the extracted feature data for efficient storage. To minimize the added compression overhead, a simple and lossy encoding is employed. However, this results in data errors when we decompress the feature data for eventual classification, which, in turn, limits the accuracy we can achieve. To overcome this problem, we retrain the classification model in an approach referred to as data-driven hardware resilience (DDHR) [6], to statistically compensate for these data errors and restore the detection performance to that of a baseline system without feature data compression. The specific contributions of this paper are as follows.

1) The concept of statistical error compensation for overcoming feature data errors from compression loss is presented and analyzed for the potential it holds in addressing memory operations. We find that substantial opportunity for compression on feature data opens up, namely three times more with statistical error compensation than without it, in our demonstrated system.

Manuscript received July 24, 2017; revised November 7, 2017; accepted February 5, 2018. This work was supported in part by NSF under Grant CCF-1253670 and in part by the Center for Future Architectures Research, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA. *(Corresponding author: Yinqi Tang.)*

The authors are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: yinqit@princeton.edu; nverma@princeton.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2808104

2) An architecture for pedestrian detection is developed whereby the use of lossy compression is exploited toward mitigating expensive memory operations. However, performing compression over a block of feature data necessitates additional buffering, which introduces additional memory overhead besides the compression and decompression operational overheads. In order to minimize the overheads, we perform architectural analysis and employ a hardware-friendly compression approach. Ultimately, the demonstrated architecture achieves four times reduction in memory energy.
3) Our proposed memory compression architecture is implemented using field-programmable gate array (FPGA). For improving detection accuracy, we use boosted decision trees on high-dimensional aggregated channel features (ACFs) [7]. Furthermore, multiscale detection is supported to increase accuracy, and the computations involved are accelerated using the method of feature pyramid approximation [8]. The system can run at 40 video graphics array (VGA) frames per second (fps) and achieves high detection accuracy (a log-average miss rate of 42%) on the Caltech Pedestrian dataset [9]. Compared to a baseline system without feature compression, the log-average miss rate is increased by just 1%, thanks to statistical error compensation. The 4× memory energy reduction by feature data compression yields 1.7× overall energy reduction for the system.

The remainder of this paper is organized as follows. Section II presents background on pedestrian detection systems, DDHR for error compensation, and memory data compression in machine learning systems. We introduce the pipeline of our pedestrian detection system in Section III. In Section IV, we describe the details of hardware implementation and present results in Section V. We conclude this paper in Section VI.

II. Background

A. Pedestrian Detection Systems

Pedestrian detection is a challenging problem in object detection due to the variability in human appearance, which can be caused by poses, clothing, lighting, scales, and occlusions. In general, pedestrian detection consists of two stages: feature extraction and classification. As shown in Fig. 1, we slide a detection window of specific size over the input image at a specific stride. For each location, the corresponding features are extracted. Then, a pretrained classification model is used to determine whether the location contains pedestrians or not.

In the literature, considerable progress has been made on both feature design and classification method to improve the detection performance. For the former, there are Haar-like features [10], [11], scale-invariant feature transform [12], shape features [13], [14], histogram of oriented gradients (HOGs) [15], and so on. For the latter, there are template matching [10], support vector machine (SVM) [11], cascade classifiers based on adaptive boosting (AdaBoost) [16], neural networks [17], [18], and so on. Recently, deep neural networks (DNNs) and convolutional neural networks (CNNs) have attracted great interest because of their high performance in object detection applications, including pedestrian detection [19]–[21]. However, the major challenge with DNNs/CNNs is their high energy consumption primarily due to large memory footprint for neural weights [22], restricting continuous operations, and always-on detection in energy constrained embedded vision systems. For example, the popular Faster R-CNN [23] detection pipeline based on a VGG-16 [24] neural network model consumes 117.7 J per 1242 × 375 image on a graphics processing unit (GPU) [25]. Even smaller models such as AlexNet [26] have shown to consume 49.3 J/image [25], which is unsuitable for energy constrained systems; indeed, studies oriented around highly specialized implementations [27] have specifically suggested that the energy of feature extraction via hardware-accelerated CNNs is 311× to 13 486× more than that of hardware for HOG features. Therefore, given the high energy cost of DNNs/CNNs, an architecture for detection systems that is emerging involves the use of low energy, but coarser, detectors that provide continuous operations, to selectively enable DNN/CNN-based detectors with reduced duty cycle [28]. In this way, high overall system performance can be achieved at low overall system energy.

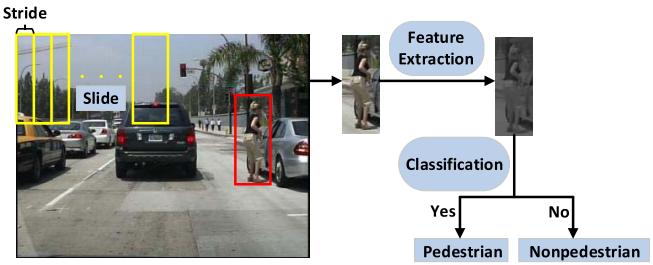


Fig. 1. Illustration of sliding window method for pedestrian detection. Image is taken from the Caltech Pedestrian dataset.

For such low-energy detectors, HOG features in combination with linear SVM or AdaBoost classification method have become popular. This is primarily because of the moderate computational cost of HOG features and their high performance in object detection. There have been many GPU solutions proposed to accelerate HOG features computation and classification [29], [30], as well as a variety of FPGA and application-specific IC (ASIC) solutions providing even greater speed and energy efficiency. Some representative implementations are listed in Table I.

Though HOG features have provided a compelling balance between accuracy and computational complexity for embedded systems, the increasing scope of applications has demanded higher accuracy. To achieve this, applications have begun to exploit combinations of features, in order to more robustly represent relevant variances in the image data [41]. ACF is an example of this, which is employed in our implementation. However, such feature representations are substantially larger (e.g., ACFs typically have 32× more features than HOG for the same size of detection window). This necessitates larger embedded memory, which dominates energy and throughput

TABLE I
PERFORMANCE SUMMARY OF VARIOUS FPGA AND ASIC IMPLEMENTATIONS FOR HOG-BASED PEDESTRIAN DETECTORS

FPGA Implementation				
Source	Platform	Scales	Resolution	FPS
[31]	Xilinx Virtex-5	3[1]	320×240	38
[32]	Xilinx Virtex-5	1	320×240	62.5
[33]	Xilinx Virtex-6	1	640×480	60
[34]	Altera Cyclone IV	1	800×600	72
[35]	Xilinx Virtex-6	13[2]	1024×768	12.7
[36]	Xilinx Virtex-5	18[3]	1920×1080	64
[37]	Xilinx Virtex-6	34[4]	640×480	68.2
ASIC Implementation				
Source	Technology	Scales	Resolution	FPS
[38]	65nm CMOS	1	1920×1080	30
[39]	45nm SOI	12	1920×1080	60
[40]	65nm CMOS	12	1920×1080	30

[1]This is an estimated value based on the total number of detection windows per image, the size of detection window, scale factor, and window stride given in this paper. Also, image scaling needs to be finished before sending the image data to the accelerator for pedestrian detection.

[2]Image scaling is performed using GPU in this implementation.

[3]This is based on time multiplex approach across three successive frames [36]. So effectively only six scales are detected for each frame.

[4]For this implementation, the gradient magnitude and orientation values at 34 scales are precalculated before sending to FPGA memory.

limitations. Further, multiscale detection for enhancing invariance to size due to near versus far objects leads to more feature data for the given image and worsens the problem. This motivates the focus in this paper to perform compression on the large amount of resulting feature data, in order to reduce the memory footprint, as well as the memory accesses and the energy per access. To minimize the overhead of such compression, a lossy encoding is used, along with statistical error compensation to restore accuracy.

B. Data-Driven Hardware Resilience

The basic form of statistical error compensation employed is called DDHR. Conventionally, application data give rise to distributions in a feature space. Then, a machine learning model (e.g., classification decision boundary) is learned by using data representative of the corresponding distributions. DDHR views nonidealities in the system as causing data errors, which can simply be considered as perturbing the original distributions. Therefore, a new model, referred to as an *error-aware model*, can be learned using representative error-affected data [6]. As long as the resulting distributions preserve mutual information with the class membership [6], high detection performance can be maintained even in the presence of severe errors. Previous work in DDHR was in the context of hardware faults (stuck-at-1/0 faults). In this paper, we leverage DDHR toward an architecture that addresses expensive memory operations by enabling aggressive memory data compression.

C. Memory Data Compression

Given the dominance of memory operations in machine learning systems, many works have begun to explore memory data optimization. The optimization can be applied either on application data (features) or the machine learning models.

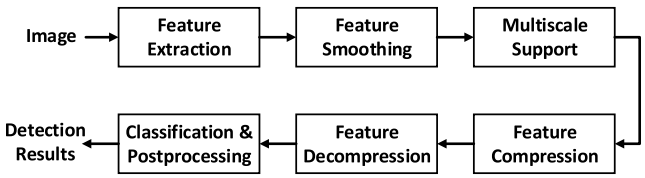


Fig. 2. High-level pipeline of the pedestrian detection system.

For the former, Moons and Verhelst [42] and Chen *et al.* [43] address energy-intensive data movement to/from off-chip memory, by exploiting Huffman-based compression and run-length coding on image data, respectively. For the latter, Lee and Verma [44] explore lossy compression on the support vectors in an SVM model, and several works reduce the model size of DNNs/CNNs [45]–[47].

However, these works employ either the lossless compression methods or strong machine learning models (DNNs/CNNs), which substantially provide implicit error tolerance for fine-tuning the architectures. The exception is [44], where lossy compression is applied to the SVM model. We focus on an even simpler machine learning model (decision trees) for low energy, and this makes feature data accessing the primary concern, and therefore, the focus for memory data compression. The use of a simpler machine learning model in conjunction with aggressive lossy compression makes statistical error compensation critical for restoring detection accuracy.

III. DETECTION PIPELINE

This section describes the pipeline of the detection system. As shown in Fig. 2, the key blocks include feature extraction, feature smoothing, multiscale support, feature compression/decompression, classification, and postprocessing.

A. Feature Extraction

For an input image I, a corresponding channel feature is a linear or nonlinear transformation of I [48]. Aggregating different channel features often increases accuracy thanks to heterogeneous information of the image. As shown in [7], the following three types of channel features, together referred to as ACF, prove both simple and effective for pedestrian detection: 1) LUV color channels; 2) a gradient magnitude channel; and 3) oriented-gradient-histogram channel. ACF extraction is a pixelwise computation. LUV color channels correspond to the transformation of image pixels from raw RGB color space to LUV color space, which requires a highly nonlinear computation. We approximate this transformation using a lookup table (LUT) to make it hardware efficient. As a result, three LUV color channels are extracted. The next two types of channel features are derived from the LUV color channels themselves. To do this, the horizontal gradient $L_x(x, y)$ and vertical gradient $L_y(x, y)$ of each LUV color channel are computed using the following equation:

$$\begin{aligned} L_x(x, y) &= L(x + 1, y) - L(x - 1, y) \\ L_y(x, y) &= L(x, y + 1) - L(x, y - 1) \end{aligned} \tag{1}$$

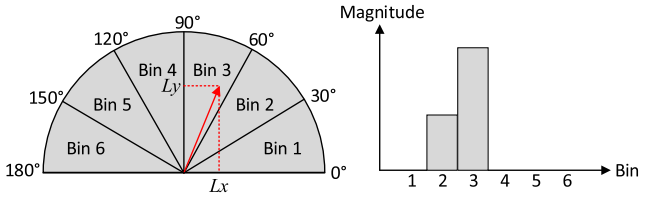


Fig. 3. Illustration of computing the oriented-gradient-histogram channels.

where $L(x, y)$ denotes the feature value of each LUV color channel at location (x, y). Then, for each pair of $L_x(x, y)$ and $L_y(x, y)$, the magnitude $M(x, y)$ is computed as follows:

$$M(x, y) = \sqrt{L_x(x, y)^2 + L_y(x, y)^2} \tag{2}$$

where the COordinate Rotation DIgital Computer (CORDIC) algorithm [49] is used to calculate the square root. To obtain the gradient magnitude channel, the maximum gradient magnitude-squared value from among the LUV color channels is selected via a comparator, and the square root is computed only for the selected value.

To compute the oriented-gradient-histogram channels, the gradient orientation $\theta(x, y)$ of the LUV color channel selected to have maximum gradient magnitude is computed using the following equation:

$$\theta(x, y) = \arctan \frac{L_y(x, y)}{L_x(x, y)} \tag{3}$$

where the CORDIC algorithm is employed to calculate the arctangent. Based on gradient orientations, the selected gradient magnitudes are assigned to different orientation bins. Six bins are employed in our implementation, which are equally spaced from 0° to 180°. For each location (x, y), its gradient magnitude is placed into the two nearest orientation bins using linear interpolation, as shown in Fig. 3 (where the gradient magnitude in bin 3 is larger than that in bin 2, and the other four bins are assigned zeros). In total, for an image of dimensionality $h \times w \times 3$ (due to RGB pixels), the extracted ACFs have a dimensionality of $h \times w \times 10$. The 10 features per pixel correspond to three LUV color channels, one gradient magnitude channel, and six oriented-gradient-histogram channels.

B. Feature Smoothing

Fine-scale smoothing of ACFs helps to increase the detection accuracy by filtering some feature noise. We apply a triangle filter with mask $\left(\frac{1}{4} \ \frac{1}{2} \ \frac{1}{4}\right)$ on the extracted ACFs in both the horizontal and vertical directions, as given in the following equation:

$$F'(x, y) = F(x, y) * M \quad M = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{4}$$

where $F(x, y)$ denotes the original ACF at location (x, y) after feature extraction, $F'(x, y)$ is the smoothed ACF, and M is the equivalent filter mask combining the triangle filters in both directions. As seen, feature smoothing is computationally simple and only involves add and shift operations. Experiments show that the detection accuracy can be increased by about 5% after performing feature smoothing.

TABLE II
DETAILS OF COMPUTING FEATURES FOR DIFFERENT SCALES

Scale	Approximation	Approximation Source
640×480	No	-
544×408	Yes	640×480
448×336	Yes	640×480
384×288	Yes	640×480
320×240	No	-
256×192	Yes	320×240
224×168	Yes	320×240
192×144	Yes	320×240
160×120	No	-
128×96	Yes	160×120
96×72	Yes	160×120

C. Multiscale Support

Multiscale detection is required because objects of interest may be various sizes, for instance, due to whether they are close by or at a distance in the image. To balance between detection accuracy and computational cost, our system adopts 11 scales for detection. For efficient computation of features at different scales, the feature pyramid approximation method is adopted [8]. Rather than explicitly computing from raw RGB values, the features for different scales are approximated from those computed for a particular scale using bilinear interpolation, followed by scaling. The following equation shows the concept:

$$\frac{f_\Omega(I_{S_1})}{f_\Omega(I_{S_2})} \approx \left(\frac{S_1}{S_2}\right)^{-\lambda_\Omega} \tag{5}$$

where I_{S_1} and I_{S_2} denote images of scale S_1 and S_2, respectively, f_Ω is the averaged feature value of all pixels in the image scale for channel Ω, and λ_Ω is the scaling factor corresponding to channel Ω. Each ACF channel has its own scaling factor, which is obtained from off-line training [8]. Such an approximation is found to be accurate for up to one octave of scaling (the set of scales up to half of the largest scale). Thus, explicit computation is required roughly every octave. In the demonstrated system, ACFs are explicitly computed for three scales and approximated for eight additional scales. The details are shown in Table II.

D. Feature Compression and Decompression

In a typical vision system, image data (e.g., from an active-matrix imager) are available row by row. However, detection is performed by scanning a window over the image, as shown in Fig. 1. In our detection system, a window of size 24×48 is employed with a stride step of 4 in both horizontal and vertical directions. This implies the need to buffer at least 48 rows of features before classification, necessitating a large and energy-intensive embedded memory for accessing. There exist some on-the-fly approaches performing partial classification computations on each row of features, to reduce the total buffering required [38], [39]. However, these apply to classification algorithms involving regular computation across the rows (e.g., SVM); such algorithms are generally found to be both computationally more expensive, with multiplication operation

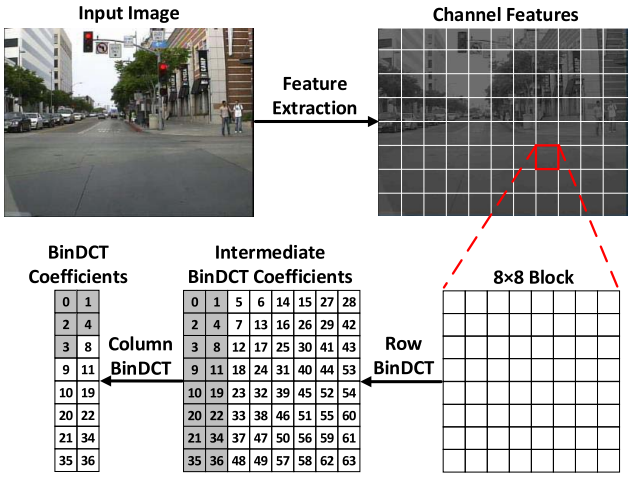


Fig. 4. Illustration of compression approach based on BinDCT, shown for one type of channel features for simplicity. The numbers in BinDCT coefficient matrix denote the zig-zag scan order from low frequency to high frequency, with the gray-marked coefficients stored for compression. Image is taken from the Caltech Pedestrian dataset.

applied to every feature, and less accurate than algorithms such as boosted decision trees, which are thus adopted in our implementation, but require irregular computational patterns across the rows.

Therefore, to overcome the large cost of buffering, we perform memory data compression on the features of all scales to reduce the memory energy. Compression has two benefits on energy. First, the size of memory is reduced, lowering the energy and delay per access [4]. Second, fewer total memory accesses are required, which further reduces energy and delay. However, feature compression introduces two additional overheads: 1) the computational cost of encoding/decoding and 2) additional buffering cost, since compression is often performed on a block of data across several rows.

To minimize the computational overhead, we employ a simple compression approach based on binary discrete cosine transform (BinDCT) [50], which is an approximation of the traditional 8×8 DCT. BinDCT involves no multiplications, only additions and shifts. This makes it efficient for hardware implementation with performance close to that of the original DCT. As shown in Fig. 4, BinDCT is performed on each 8×8 feature block. Feature accessing and decompression are only performed as needed for classification. However, minimally eight rows of buffering are thus required. BinDCT results in 64 coefficients. To balance between the high compression ratio and detection accuracy, we fix the number of selected low-frequency BinDCT coefficients to 5 (following a zig-zag scan order as shown) based on experiments, which will be described further in Section V. In addition, BinDCT is separated into row BinDCT and column BinDCT. In this way, only two columns of intermediate BinDCT coefficients containing the selected coefficients, rather than all eight columns corresponding to the feature block, need to be buffered.

The use of 5 BinDCT coefficients yields over $12\times$ compression on the feature data. However, BinDCT compression is lossy and requires retraining of the classifier to restore performance. To perform classification, the stored BinDCT coefficients are read from memory and then decompressed to the 8×8 feature block via an inverse BinDCT. Eighteen feature blocks need to be decompressed in this way and buffered to form the detection window (24×48) for classification.

E. Classification and Postprocessing

For classification, a variant of boosted decision trees, called soft cascade is used. As proposed in [51], soft cascade employs an early termination criterion. In the pedestrian detection system, we perform classification using 2048 decision trees of depth-3 on extracted ACFs of each detection window. The decision trees are accessed sequentially for boosting. After evaluating each tree, an accumulated detection score is recomputed and compared against a threshold. If the detection score is smaller than the threshold, further evaluations are terminated and negative detection is declared, thus saving computations and achieving fast detection (i.e., from experiments, about 90% of the detection windows are classified as negative using fewer than 10 decision trees).

In terms of hardware, the system employs four classification engines to perform detection for the different scales in parallel. To balance the computational load, the first classifier is applied to the largest scale, the second classifier is applied to the next two largest scales, the third classifier is applied to the next three largest scales, and the fourth classifier is applied to the remaining five scales.

After classification on every detection window at every scale, there can exist situations where a single pedestrian has been detected from multiple feature vectors, thus resulting in many duplicate detections. To resolve this, we employ the method of non-maximum suppression (NMS) to suppress multiple nearby positive detections, similar to [48]. Specifically, each positive detection is associated with a confidence score after classification, which denotes the probability of matching with ground truth. If the overlap area of two detections is sufficiently large according to the PASCAL criterion [52], the window with smaller confidence score will be discarded.

F. Overall Architecture and Buffering Requirements

Fig. 5 shows the overall architecture of the pedestrian detection system. In Fig. 5 (top), feature extraction, feature smoothing, and multiscale support are shown. In Fig. 5 (bottom), feature compression/decompression, classification, and postprocessing, following multiscale support, are shown. All of the 11 scales are buffered in random access memory (RAM) before classification so that, during horizontal and vertical striding over the image, most features of a detection window are available without recomputation. As mentioned, additional buffering is required for BinDCT compression, with buffering of the intermediate results optimized through the use of row and column BinDCT. In addition to the proposed system shown, we also implement a baseline system without feature compression/decompression (and all of the associated buffering) for comparison (results in Section V); the red dashed boxes highlight the blocks that are different between the proposed and baseline systems.

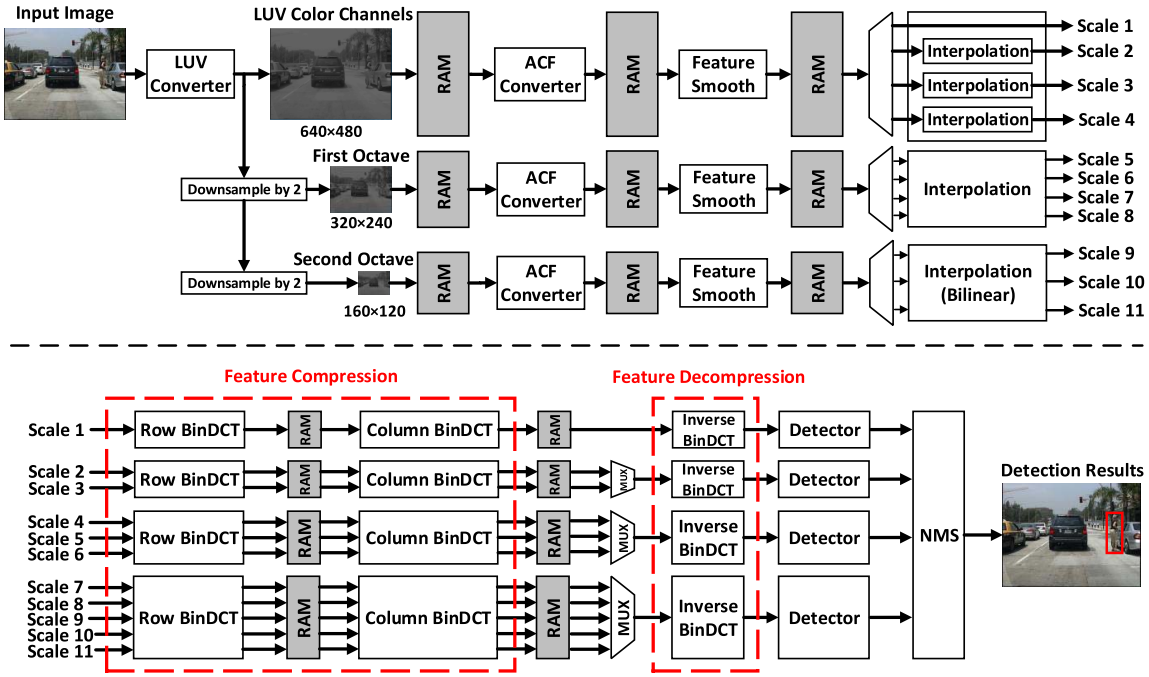


Fig. 5. Overall architecture of the pedestrian detection system. Images are taken from the Caltech Pedestrian dataset.

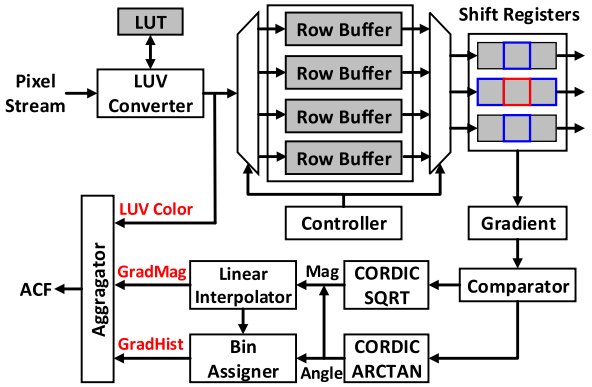


Fig. 6. Block diagram of feature extraction.

IV. Hardware Implementation

This section provides further details of the system hardware implementation, including feature extraction, bilinear interpolation, feature compression/decompression, and classification blocks.

A. Feature Extraction

Fig. 6 shows the block diagram of feature extraction. RGB image pixels stream in row by row for feature computation. An LUV converter then transforms these pixels to LUV color space, via an LUT approximation. Next, four row buffers are used to calculate the horizontal and vertical gradients for each LUV color channel of each pixel. Three row buffers are used for the gradient computations (as gradient computation requires LUV color channels of the four adjacent pixels), and the fourth is used to buffer incoming data, in a circulating manner by the controller. For every clock cycle, the LUV color channels from the three rows are simultaneously read from the row buffers into output shift registers, with each shift register having a depth of three (i.e., a total of nine register locations is needed, each storing three LUV color channel values). As shown, the LUV color channels of the blue marked pixels are used for computing the gradients of the red marked pixel.

Following gradient computations, a comparator selects the maximum gradient magnitude-squared value from among the LUV color channels, and then a CORDIC unit takes the square root, as required for the gradient magnitude channel (GradMag). Furthermore, for the LUV color channel selected to have a maximum gradient magnitude, the bin assigner, in combination with another CORDIC unit and a linear interpolator, determines the gradient orientation and puts the gradient magnitude to different orientation bins, as required for the oriented-gradient-histogram channels (GradHist). Both CORDIC units employ pre- and postscale schemes to extend the range of convergence [53]. Computations inside the CORDIC units are pipelined to enhance throughput. With continuous image data, the feature extraction pipeline can output ACF for a new pixel every clock cycle. A similar structure is employed for feature smoothing, with the difference that the features stored in all nine shift register locations are used for one convolution computation with the filter mask.

B. Bilinear Interpolation

Bilinear interpolators are utilized to obtain ACFs for different scales. As shown in Table II, a particular scale is typically used to approximate several others. To reduce memory access energy and increase the speed of feature generation, the approximated scales are computed simultaneously, thus accessing source ACFs only once. Fig. 7 shows the

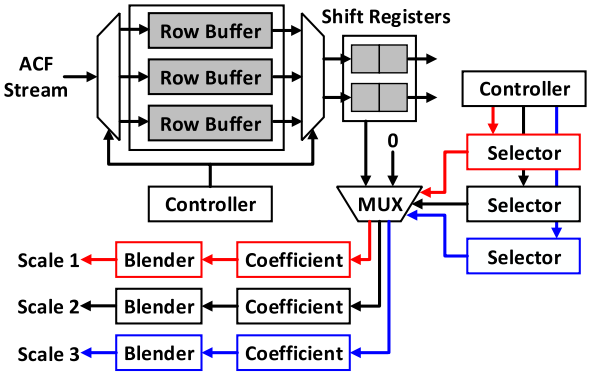


Fig. 7. Block diagram of bilinear interpolation, showing the generation of features for three additional scales, as an illustration.

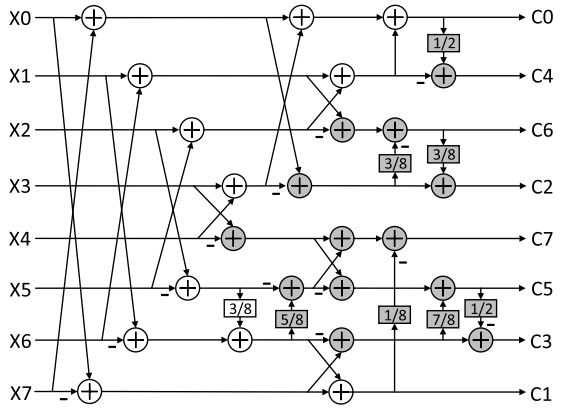


Fig. 8. Computational structure of row BinDCT for each of the eight rows. Numbers in rectangular boxes denote multipliers.

implementation, wherein three row buffers are used. Similar to the feature extraction architecture, one is used to buffer incoming source ACFs and the other two are used to readout ACFs to output shift registers, in a circular manner by the controller. There are in total four shift register locations over the two different rows, all of which are used to calculate the interpolated features. Each approximated scale has a selector, which determines whether the current ACFs in shift registers are necessary to calculate the interpolated ACF of the corresponding scale. If not necessary, further steps are gated to reduce computational cost. Otherwise, each scale will use its own coefficient unit and blender to complete the bilinear interpolation. The coefficient unit calculates the coefficients associated with the ACFs in each shift register location, and the blender computes the final interpolated ACF. The computations in both the coefficient unit and the blender are pipelined to enhance throughput.

C. Feature Compression

As mentioned in Section III, BinDCT is separated into row BinDCT and column BinDCT for feature compression, to require buffering of only two columns of intermediate BinDCT coefficients. Fig. 8 shows the computational structure of row BinDCT applied to each of the eight rows. Since only five coefficients are retained for compression, the gray marked

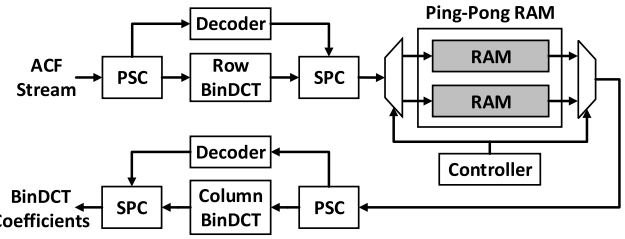


Fig. 9. Block diagram of feature compression.

blocks are omitted compared with the original implementation in [50]. For column BinDCT, the computational structure is the same, with similar optimizations.

Fig. 9 shows the block diagram of feature compression. For compression over an 8×8 feature block, the buffering required is implemented by a ping-pong RAM structure. Each RAM contains eight rows of intermediate BinDCT coefficients. One RAM is accessed for column BinDCT computation, and the other is used to receive incoming intermediate BinDCT coefficients. In addition, since ACFs yield 10 channel features per pixel, a parallel-to-serial converter (PSC) and serial-to-parallel converter (SPC) are used around row/column BinDCTs, with a channel decoder for communication. In this way, we can calculate the BinDCT coefficients channel by channel, thus reducing hardware cost. The computations inside row/column BinDCTs are pipelined to enhance throughput.

D. Feature Decompression and Classification

Fig. 10 shows the block diagram of feature decompression and classification, where four detectors are employed to detect different scales in parallel. The computed BinDCT coefficients of all scales are stored in memory. As mentioned in Section III, for load sharing across the four classifiers, some classifiers are associated with multiple scales. The arbitrator controls the scale that is processed when the corresponding classifier is not busy. Larger scales are given higher priority because they have more detection windows requiring processing. In this way, downstream pipeline stalls are minimized. The coefficients of the selected scale are read from memory and an inverse BinDCT is performed with the support of PSC, SPC, and channel decoder. As only five BinDCT coefficients are stored for a feature block, the other 59 coefficients are set to zero for decompression. This enables simplification of the computations for inverse BinDCT, similar to BinDCT described earlier. Then, a small classification buffer is used to store the decompressed features corresponding to an entire detection window. This avoids reaccess and redecompression of features in the overlapping area of successive detection windows during horizontal striding. The boosted decision tree classifier is applied to the decompressed features. As shown, the pretrained classification model is stored in memory and accessed through an address decoder, which exploits the tree structure to decode memory locations for accessing. Next, a comparator is used to evaluate the accumulated detection score against a threshold, in order to determine early termination. All positive detections are finally processed by the NMS unit, to generate the

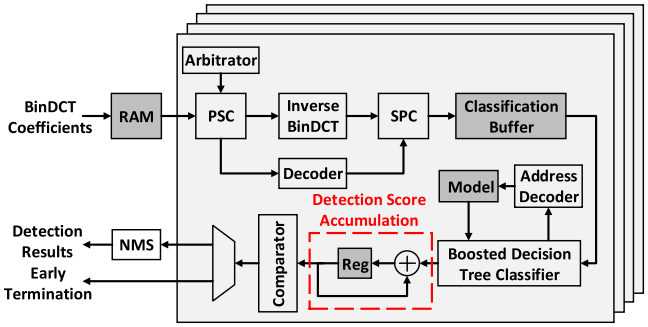


Fig. 10. Block diagram of feature decompression and classification. Four detectors, shown as large boxes in gray, work in parallel for different scales.

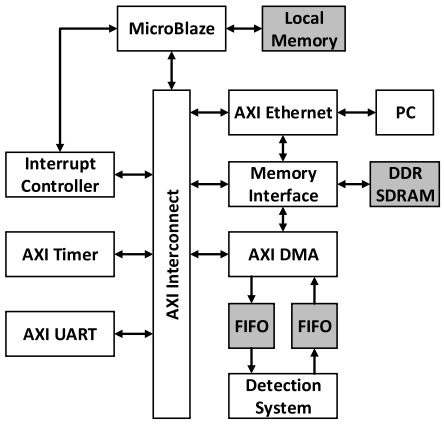


Fig. 11. Architecture of FPGA system.

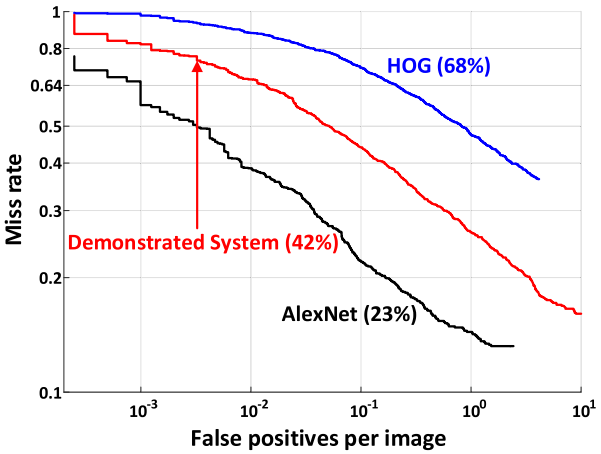


Fig. 12. Detection accuracy on the Caltech Pedestrian dataset. The curves for HOG and AlexNet are extracted from the Caltech Pedestrian website. (http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/datasets/USA/res).

detection results for the current image. The structures of the four parallel detectors are similar, with the exception of the arbitrators, which manage different numbers of scales in each case (there is no arbitrator in the detector for the largest scale since it only processes the 640×480 scale).

V. System Evaluation

In this section, we first describe the details of system implementation on an FPGA. Then, we characterize the implementation and its performance in terms of various metrics, to evaluate its merits.

A. FPGA Implementation

The pedestrian detection system is developed using Verilog RTL and implemented on a Xilinx Virtex-7 VC707 evaluation board, using the Vivado (v2016.2) tool set. Because of the early termination approach employed in the boosted decision tree classifier, each image requires slightly different time for detection, but the time taken is observed to be 3 million clock cycles for a typical VGA image. With a clock frequency of 120 MHz, the resulting detection speed of the system is roughly 40 VGA fps.

To characterize functionality, we feed images to the FPGA implementation from a personal computer (PC). The overall architecture based on the FPGA-embedded MicroBlaze processor is shown in Fig. 11. Image data are received from the PC via Ethernet, stored into a double data rate synchronous dynamic RAM (DDR SDRAM), and then provided to the detection system implementation via direct memory access (DMA). A ping-pong RAM structure is used to receive incoming image data from the PC and readout image data to the detection system. Since the detection system is running at 120 MHz, while the DMA is running at 175 MHz, an asynchronous first-in first-out (FIFO) is used as the bridge for data transmission between different clock domains. The detection results are stored into DDR SDRAM again using DMA, with the support of another asynchronous FIFO, and finally sent back to the PC via Ethernet. The C-based embedded software running on MicroBlaze controls the system, mainly for Ethernet communication with PC based on lightweight IP [54] and DMA configuration. All peripherals, including Ethernet, DMA, memory interface, interrupt controller, timer, and universal asynchronous receiver/transmitter, are managed by the MicroBlaze processor through advanced extensible interface interconnect [55].

B. Detection Accuracy

We employ the Caltech Pedestrian dataset [9] to evaluate the accuracy of our detection system. To maintain high detection accuracy, two-byte fixed-point representation is utilized for each ACF channel. The classification model is trained using a hardware-equivalent fixed-point MATLAB implementation based on a toolbox [56]. For evaluation, we adopt the *full-image* methodology [41], which is more meaningful than the traditional *per-window* methodology. Fig. 12 shows the detection accuracy for three algorithms, by plotting curves of miss rate against false positives per image (FPPI). Miss rate denotes the proportion of pedestrians that are not detected, while FPPI denotes the average number of nonpedestrian instances that are incorrectly classified as pedestrians in each image. A lower curve indicates better detection accuracy. We use log-average miss rate [41] to summarize the detection performance under different FPPI rates. As seen, our system achieves a log-average miss rate of 42%. Compared to HOG features, employed in most existing low-power pedestrian

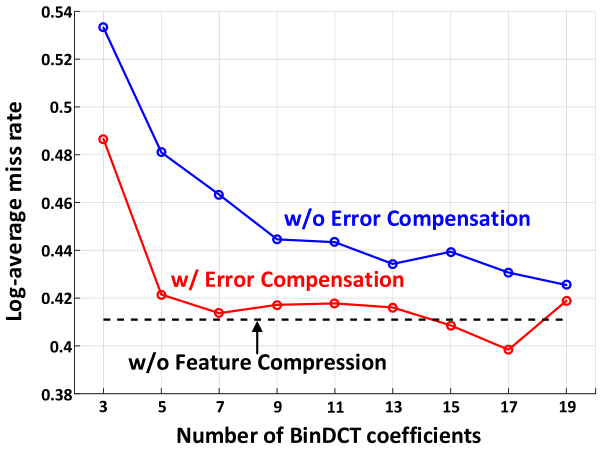


Fig. 13. Statistical error compensation performance on log-average miss rate with different selected numbers of BinDCT coefficients. Black horizontal dashed line: log-average miss rate without coefficients down selection (i.e., all 64 coefficients of one feature block are kept).

detection accelerators, the accuracy is better by 26% (the HOG performance shown is based on multiple detection scales [15], while many accelerator implementations perform single-scale detection, resulting in lower detection accuracy). For reference, a state-of-the-art algorithm [21] based on AlexNet convolutional neural network is also shown. This exhibits superior performance with 23% log-average miss rate but has a much higher energy consumption, as described in Section II. The proposed low-energy detector may be used as a coarse estimator to reduce the duty cycle of such a system.

C. Performance of Statistical Error Compensation

Fig. 13 shows the log-average miss rate on the Caltech Pedestrian dataset versus the number of BinDCT coefficients selected for storing in each 8×8 feature block, both with and without statistical error compensations (via DDHR). The log-average miss rate of the baseline system without feature compression (i.e., without coefficients down selection) is 41.1%, as denoted by the black horizontal dashed line. Fewer coefficients lead to larger compression ratio, but worse detection performance due to more information loss. As seen, without error compensation, detection accuracy begins to decrease when we select 17 BinDCT coefficients. With error compensation, high detection accuracy can be maintained even when only five coefficients are chosen (fixed in our implementation). Thus, error compensation enables three times further compression on feature data with minimal degradation in detection performance (about 1% accuracy compared to baseline accuracy without compression).

D. Advantages of Feature Compression

As mentioned in Section III, feature compression reduces both memory footprint and memory accesses, thus leading to lower energy consumption and faster detection speed. Table III lists the total buffering requirement by block and the corresponding memory accesses estimated per image, both with and without feature compression. The buffering requirements are measured by the number of 36-kbit block RAMs (BRAMs)

TABLE III

BUFFERING REQUIREMENT BY BLOCK AND ESTIMATED MEMORY ACCESSES PER IMAGE

	w/o Feature comp.		w/ Feature comp.	
Block	BRAMs	Accesses	BRAMs	Accesses
ACF calculation	14	1.6M	14	1.6M
ACF smoothing	30	1.6M	30	1.6M
Interpolation	28.5	1.2M	28.5	1.2M
Feature comp.	-	-	85	0.5M
Classification[1]	840	7.1M	77.5	0.6M

[1]Here, we do not consider the classification buffer, as its size is relatively small compared to that of buffers storing ACFs or BinDCT coefficients.

TABLE IV

FPGA RESOURCE UTILIZATION FOR THE DETECTION SYSTEMS WITH AND WITHOUT FEATURE COMPRESSION

	w/o Feature comp.		w/ Feature comp.	
Resource	Used	Percentage	Used	Percentage
Slice LUTs	72990	24.04%	105670	34.81%
Slice Registers	58881	9.70%	123116	20.28%
36Kb BRAMs	1018.5	98.88%	341	33.11%
DSPs	633	22.61%	633	22.61%

utilized within the FPGA. As seen, without feature compression, we need 840 BRAMs to buffer ACFs for classification. However, with feature compression, only 77.5 BRAMs are needed to buffer BinDCT coefficients. In addition, the total memory accesses for forming detection windows are reduced from 7.1 to 0.6 M, which leads to lower energy and higher detection speed. Based on simulations, the system without feature compression can detect only 32 VGA fps (versus 40 fps with feature compression) at 120 MHz. The overheads of feature compression include 85 BRAMs for buffering intermediate BinDCT coefficients and the corresponding 0.5 M memory accesses.

Table IV lists the FPGA resource utilization for the systems with and without feature compression. For the system using feature compression, more LUTs and registers are required due to the computational (compression/decompression) overheads. However, the total number of BRAMs is greatly reduced thanks to feature compression, from 1018.5 to 341 (about three times).

Finally, we use the power estimator in the Vivado tool set to estimate the power consumption of both systems. For comparison, we keep the throughput of both systems at 32 fps on VGA images, resulting in 100-MHz clock frequency for the system with feature compression and 120 MHz for the system without feature compression. We use the value change dump simulation files to set the toggle rates of all signals. The results are shown in Table V. As seen, the memory energy in the system with feature compression is reduced by $4\times$, which leads to $1.7\times$ overall energy reduction, including all overheads for feature compression.

E. Comparison With Other Systems

For comparison with state of the art, we employ an FPGA-based design of pedestrian detector implemented in [34], chosen because of its implementation of a complete detection pipeline and detailed power reporting [57]. Table VI shows the results. As seen, by employing ACFs in combination

TABLE V

POWER BREAKDOWN FOR THE SYSTEMS WITH AND WITHOUT FEATURE COMPRESSION (AT 32 fps ON VGA FRAMES)

	w/o Feature comp.	w/ Feature comp.
On-chip component	Power (Watt)	Power (Watt)
Clocks	0.194	0.225
Slice Logic	0.032	0.045
Signals	0.041	0.057
BRAMs	0.504	0.128
DSPs	0.007	0.007
I/O	< 0.001	< 0.001
Static Power	0.339	0.209
Total	1.117	0.671

TABLE VI

PERFORMANCE COMPARISON BETWEEN THE PROPOSED SYSTEM (WITH FEATURE COMPRESSION) AND THE IMPLEMENTATION IN [34]

	[34]	This Work
Platform	Altera Cyclone IV	Xilinx Virtex-7
Image resolution	800×600	640×480
Scales	1	11
Total pixels	0.48M	1.02M
Window size	64×128	24×48
Window stride	8	4
Total windows	5580	51841
Feature	HOG	ACF
Features per window	3780	11520
Classifier	SVM	Boosted decision tree
Frequency (MHz)	40	120
Throughput (fps)	72 (single-scale)	40 (multiscale)
Miss rate	$> 68\%$	42%
Power (Watt)	0.197	0.828
Energy per window (nJ)	490.34	399.30

with a boosted decision tree classifier to detect 11 scales for each image, the accuracy is significantly improved (lower miss rate by over 26%) beyond that achieved by single-scale detection [34]. Though the throughput of our system is lower due to over nine times more windows for multiscale detection, the energy per detection window is significantly lower (399 versus 490 nJ) thanks to the architectural optimizations proposed.

VI. CONCLUSION

Memory operations introduced by high-dimensional feature data pose both energy and speed bottlenecks in embedded vision systems. Such systems increasingly employ machine learning algorithms that are showing substantial promise for statistical error compensation. In this paper, we propose a hardware accelerator for pedestrian detection. The detection pipeline is based on ACF extraction and boosted decision tree classification, which shows higher detection accuracy than the traditional method based on HOG features and SVM/AdaBoost classification. However, ACFs have higher dimensionality than HOG features, thus exacerbating memory operations. To overcome this, we employ lossy compression based on BinDCT to reduce memory footprint as well as memory accesses, with minimal overhead for encoding and decoding the feature data. Then, to overcome errors due to lossy compression, we employ classifier retraining as statistical error compensation, restoring the system performance. The detection system is demonstrated on an FPGA, achieving a throughput of 40 fps on VGA images. The memory energy is reduced by $4\times$ with feature compression, and overall energy is reduced by $1.7\times$, considering all overheads of the architecture.

REFERENCES

[1] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 7, pp. 1239–1258, Jul. 2010.

[2] A. Whitmore, A. Agarwal, and L. D. Xu, "The Internet of Things—A survey of topics and trends," *Inf. Syst. Frontiers*, vol. 17, no. 2, pp. 261–274, Apr. 2015.

[3] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *Int. J. Comput. Aided Eng. Technol.*, vol. 6, no. 4, pp. 440–459, 2014.

[4] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[5] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *Proc. Design Autom. Conf.*, Jun. 2010, pp. 555–560.

[6] Z. Wang, K. H. Lee, and N. Verma, "Overcoming computational errors in sensing platforms through embedded machine-learning kernels," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 8, pp. 1459–1470, Aug. 2015.

[7] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 8, pp. 1532–1545, Aug. 2014.

[8] P. Dollár, S. J. Belongie, and P. Perona, "The fastest pedestrian detector in the west," in *Proc. Brit. Mach. Vis. Conf. (BMVA)*, Sep. 2010, pp. 68.1–68.11.

[9] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 304–311.

[10] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio, "Pedestrian detection using wavelet templates," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 1997, pp. 193–199.

[11] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *Int. J. Comput. Vis.*, vol. 38, no. 1, pp. 15–33, 2000.

[12] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[13] D. M. Gavrila, "A Bayesian, exemplar-based approach to hierarchical shape matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 8, pp. 1408–1421, Aug. 2007.

[14] P. Sabzmeydani and G. Mori, "Detecting pedestrians by learning shapelet features," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2007, pp. 1–8.

[15] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2005, vol. 1. no. 1, pp. 886–893.

[16] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," *Int. J. Comput. Vis.*, vol. 63, no. 2, pp. 153–161, 2005.

[17] S. Munder and D. M. Gavrila, "An experimental study on pedestrian classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 11, pp. 1863–1868, Nov. 2006.

[18] D. M. Gavrila and S. Munder, "Multi-cue pedestrian detection and tracking from a moving vehicle," *Int. J. Comput. Vis.*, vol. 73, no. 1, pp. 41–59, Jun. 2007.

[19] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, "Pedestrian detection with unsupervised multi-stage feature learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 3626–3633.

[20] W. Ouyang and X. Wang, "Joint deep learning for pedestrian detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 2056–2063.

[21] J. Hosang, M. Omran, R. Benenson, and B. Schiele, "Taking a deeper look at pedestrians," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 4073–4082.

[22] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[23] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2015, pp. 91–99.

[24] K. Simonyan and A. Zisserman. (Sep. 2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556

[25] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer. (Dec. 2016). "SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving." [Online]. Available: https://arxiv.org/abs/1612.01051

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2012, pp. 1097–1105.

[27] A. Suleiman, Y.-H. Chen, J. Emer, and V. Sze, "Towards closing the energy gap between HOG and CNN features for embedded vision," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[28] Z. Wang and N. Verma, "A low-energy machine-learning classifier based on clocked comparators for direct inference on analog sensors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 11, pp. 2954–2965, Nov. 2017.

[29] L. Zhang and R. Nevatia, "Efficient scan-window based object detection using GPGPU," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2008, pp. 1–7.

[30] H. Sugano and R. Miyamoto, "Parallel implementation of pedestrian tracking using multiple cues on GPGPU," in *Proc. IEEE 12th Int. Conf. Comput. Vis. Workshops (ICCV Workshops)*, Sep. 2009, pp. 900–906.

[31] M. Hiromoto and R. Miyamoto, "Hardware architecture for high-accuracy real-time pedestrian detection with COHOG features," in *Proc. IEEE 12th Int. Conf. Comput. Vis. Workshops (ICCV Workshops)*, Sep./Oct. 2009, pp. 894–899.

[32] K. Negi, K. Dohi, Y. Shibata, and K. Oguri, "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2011, pp. 1–8.

[33] M. Komorkiewicz, M. Kluczewski, and M. Gorgon, "Floating point HOG implementation for real-time multiple object detection," in *Proc. 22nd Int. Conf. Field Program. Logic Appl. (FPL)*, Aug. 2012, pp. 711–714.

[34] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "Architectural study of HOG feature extraction processor for real-time object detection," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2012, pp. 197–202.

[35] C. Blair, N. M. Robertson, and D. Hume, "Characterizing a heterogeneous system for person detection in video using histograms of oriented gradients: Power versus speed versus accuracy," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 3, no. 2, pp. 236–247, Jun. 2013.

[36] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "FPGA-based real-time pedestrian detection on high-resolution images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2013, pp. 629–635.

[37] X. Ma, W. A. Najjar, and A. K. Roy-Chowdhury, "Evaluation and acceleration of high-throughput fixed-point object detection on FPGAs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 6, pp. 1051–1062, Jun. 2015.

[38] K. Takagi, K. Mizuno, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "A sub-100-milliwatt dual-core HOG accelerator VLSI for real-time multiple object detection," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 2533–2537.

[39] A. Suleiman and V. Sze, "An energy-efficient hardware implementation of HOG-based object detection at 1080 HD 60 fps with multi-scale support," *J. Signal Process. Syst.*, vol. 84, no. 3, pp. 325–337, Sep. 2016.

[40] A. Suleiman, Z. Zhang, and V. Sze, "A 58.6 mW real-time programmable object detector with multi-scale multi-object support using deformable parts model on 1920×1080 video at 30 fps," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2016, pp. 1–2.

[41] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 4, pp. 743–761, Apr. 2012.

[42] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2016, pp. 1–2.

[43] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[44] K. H. Lee and N. Verma, "A low-power microprocessor for data-driven analysis of analytically-intractable physiological signals in advanced medical sensors," in *Proc. Symp. VLSI Circuits*, Jun. 2013, pp. C250–C251.

[45] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2014, pp. 1269–1277.

[46] S. Han, H. Mao, and W. J. Dally. (Oct. 2015). "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding." [Online]. Available: https://arxiv.org/abs/1510.00149

[47] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2015, pp. 2285–2294.

[48] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," in *Proc. Brit. Mach. Vis. Conf. (BMVA)*, Sep. 2009, pp. 91.1–91.11.

[49] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, 1959.

[50] T. D. Tran, "The BinDCT: Fast multiplierless approximation of the DCT," *IEEE Signal Process. Lett.*, vol. 7, no. 6, pp. 141–144, Jun. 2000.

[51] L. Bourdev and J. Brandt, "Robust object detection via soft cascade," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 2. Jun. 2005, pp. 236–243.

[52] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

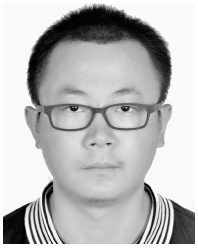
[53] J. S. Walther, "A unified algorithm for elementary functions," in *Proc. Spring Joint Comput. Conf.*, May 1971, pp. 379–385.

[54] A. Sarangi, S. MacMahon, and U. Cherukupaly, "Lightweight ip application examples," Xilinx Corp., San Jose, CA, USA, Appl. Note XAPP1026, 2014.

[55] P. Kumbhare and V. Krishna, "Designing high-performance video systems in 7 series FPGAs with the AXI Interconnect," Xilinx, Inc., San Jose, CA, USA, Appl. Note: 7 series FPGAs, XAPP741 (v1.3), Apr. 2014, pp. 1–24.

[56] P. Dollár. *Piotr's Computer Vision MATLAB Toolbox (PMT)*. Accessed: Jul. 24, 2017. [Online]. Available: https://github.com/pdollar/toolbox

[57] K. Takagi, K. Tanaka, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "A real-time scalable object detection system using low-power HOG accelerator VLSI," *J. Signal Process. Syst.*, vol. 76, no. 3, pp. 261–274, Sep. 2014.



Yinqi Tang (S'17) received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2014. He is currently working toward the Ph.D. degree at the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA.

His current research interests include high-performance and energy-efficient VLSI systems for machine learning applications, in both algorithm and hardware design aspects.

Naveen Verma (S'04–M'09) received the B.A.Sc. degree in electrical and computer engineering from The University of British Columbia, Vancouver, BC, Canada, in 2003 and the M.S. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2005 and 2009, respectively.

Since 2009, he has been with the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, where he is currently an Associate Professor. His current research interests include advanced sensing systems, including low-voltage digital logic and SRAMs, low-noise analog instrumentation and data conversion, large-area sensing systems based on flexible electronics, and low-energy algorithms for embedded inference, especially for medical applications.

Dr. Verma was a recipient or co-recipient of the 2006 DAC/ISSCC Student Design Contest Award, the 2008 ISSCC Jack Kilby Paper Award, the 2012 Alfred Rheinstein Junior Faculty Award, the 2013 NSF CAREER Award, the 2013 Intel Early Career Award, the 2013 Walter C. Johnson Prize for Teaching Excellence, the 2013 VLSI Symposium Best Student Paper Award, the 2014 AFOSR Young Investigator Award, the 2015 Princeton Engineering Council Excellence in Teaching Award, and the 2015 IEEE TRANSANCTIONS ON CPMT Best Paper Award. He is a Distinguished Lecturer of the IEEE Solid-State Circuits Society. He serves on the technical program committees for ISSCC, VLSI Symposium, DATE, and the IEEE Signal Processing Society.