A Hardware Accelerator for Real Time Sliding Window Based Pedestrian Detection on High Resolution Images

Asim Khan(✉), Muhammad Umar Karim Khan, Muhammad Bilal, and Chong-Min Kyung

Department of Electrical Engineering,
Korea Advanced Institute of Science and Technology (KAIST),
Daejeon, South Korea
{asimkhan, umar, bilalm, kyung}@kaist.ac.kr

Abstract. Pedestrian detection has lately attracted considerable interest from researchers due to many practical applications. However, the low accuracy and high complexity of pedestrian detection has still not enabled its use in successful commercial applications. In this chapter, we present insights into the complexity-accuracy relationship of pedestrian detection. We consider the Histogram of Oriented Gradients (HOG) scheme with linear Support Vector Machine (LinSVM) as a benchmark. We describe parallel implementations of various blocks of the pedestrian detection system which are designed for full-HD (1920×1080) resolution. Features are improved by optimal selection of cell size and histogram bins which have been shown to significantly affect the accuracy and complexity of pedestrian detection. It is seen that with a careful choice of these parameters a frame rate of 39.2 fps is achieved with a negligible loss in accuracy which is 16.3x and 3.8x higher than state of the art GPU and FPGA implementations respectively. Moreover 97.14 % and 10.2 % reduction in energy consumption is observed to process one frame. Finally, features are further enhanced by removing petty gradients in histograms which result in loss of accuracy. This increases the frame rate to 42.7 fps (18x and 4.1x higher) and lowers the energy consumption by 97.34 % and 16.4 % while improving the accuracy by 2 % as compared to state of the art GPU and FPGA implementations respectively.

Keywords: FPGA · Low power · Object detection · Real-time

1 Introduction

Researchers in industry and academia have been striving for accurate and real-time pedestrian detection (PD) for more than a decade owing to many commercial and military applications. Industries such as surveillance, robotics, and entertainment will be greatly influenced by appropriate application of PD. Advanced driver assistance systems (ADAS) and unmanned ground vehicles (UGV) are merely a distant dream without automated pedestrian detection. The fact that more than 15 % of traffic accidents include pedestrians [1] shows the importance of real-time pedestrian detection for the modern society [2].

Y. Shin et al. (Eds.): VLSI-SoC 2015, IFIP AICT 483, pp. 46–66, 2016.
DOI: 10.1007/978-3-319-46097-0_3

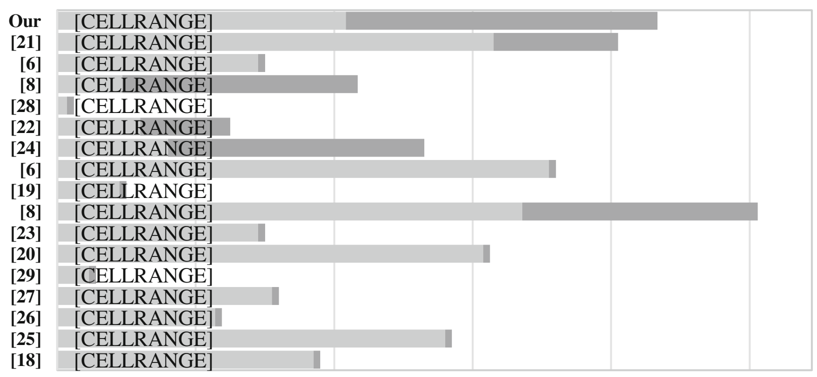
Amid numerous applications, the search for an accurate yet fast PD algorithm is ongoing. Researchers have shown great interest over the past few years in extracting diverse features from an image and finding an appropriate classification method to perform robust PD [10–14]. However, the histogram of oriented gradients (HOG) approach has proven to be a groundbreaking effort, and has shown good accuracy in various illumination conditions and multiple textured objects. Inspired from SIFT [5], the authors in their seminal paper [4] present a set of features over a dense grid in a search window. For training and classification, they used the linear support vector machine (linSVM). Their work inspired many other researchers and is still used as a benchmark PD scheme.

Although HOG was presented many years back, it is surprising to see that very few efforts have been made for an optimal hardware implementation of HOG. In fact, most of the research has been targeting pedestrian detection on a high end CPU or GPU or combination of both [24–29]. Field Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuits (ASIC) often provide better execution speed and energy efficiency as compared to GPUs due to deep pipelined architectures. Furthermore, in many embedded applications, such as surveillance, there are numerous constraints on hardware cost, speed, and power consumption. For such applications, it is more suitable to use task-specific (FPGA, ASIC) rather than general-purpose platforms. Moreover, to meet such constraints, certain parameters of the algorithm need to be tuned and an insight is required into how the change of parameters of PD affects not only the accuracy but also the hardware complexity.

Efforts have been made in the research community to either improve the accuracy of PD or reduce the hardware complexity of HOG. In [6] and [7], the computational complexity of HOG is reduced with cell-based scanning and simultaneous SVM calculation using FPGA and ASIC implementations for full HD resolution; however, the implementations use the parameters as suggested in [4]. Various hardware optimizations are presented in [15–22] for an efficient pedestrian detection system. However, for real-time PD with power and area constraints, it is imminent to find the set of parameters of HOG that provide the best compromise in terms of computational complexity and accuracy. Recently, a hardware architecture for fixed point HOG implementation has been presented [8] where the bit-width has been optimized to achieve significant improvement in power and throughput. We believe that in addition to bit-width there are other parameters which need to be optimized to provide a holistic understanding of the relationship between accuracy, speed, power, and complexity. Moreover, sliding window based pedestrian detection requires detection to be performed at multiple scales of image. It has been shown that the best detection performance can be achieved with scale factor (the ratio to scale the image after each detection) of at-least 1.09. This results in processing around 45 scales for full-HD resolution. As shown in Fig. 1, a combination of the number of scales (=45) required for maximum accuracy [8, 9] and throughput for real-time pedestrian detection at full-HD resolution has not been achieved before.

The key contributions of our work can be summarized as follows.

- We present parallel implementation of various blocks of HOG-based PD on an FPGA. Parallel implementation has been used to improve the speed of PD.



	[18]	[25]	[26]	[27]	[29]	[20]	[23]	[8]	[19]	[6]	[24]	[22]	[28]	[8]	[6]	[21]	Our
□ fps	38	57	23.8	32	5.6	62.5	30	68.2	10	72	17	13	2.4	10.4	30	64	42.7
■ scales	1	1	1	1	1	1	1	34	1	1	37	13	1	34	1	18	45

Fig. 1. Comparison of the state of the art in terms of frames per second and number of scales.

- We derive the accuracy, speed, power, and hardware complexity results of HOG-based PD with different choices of cell sizes and number of histogram bins.
- We show that by using the right choice of cell size and number of histogram bins, a significant reduction in power consumption and increase in throughput can be achieved with reasonable accuracy.
- Finally features are refined by removing insignificant gradients which results in not only improvement of throughput and power consumption but also accuracy.

This chapter is an extension of our original paper [10] including more detailed literature survey and hardware implementation details. The rest of the chapter is organized as follows. Section 2 summarizes the state of the art in PD. In Sect. 3, a brief overview of HOG is presented. The proposed hardware implementation is discussed in Sect. 4. In Sect. 5, the accuracy, speed, power, and hardware complexity results are shown for different choices of parameters and the optimal choice of parameters under given constraints is described. Section 6 concludes the work.

2 Literature Survey

Numerous efforts have been made in the past to perform PD efficiently. An extensive survey of PD schemes is given in [9]. Generally, these approaches can be classified into two categories: segmentation-based approaches [34] and sliding window-based approaches [11]. A segmentation-based approach processes the whole frame at once and extracts segments of the frame which include pedestrians. On the other hand, sliding window-based approaches divide a frame into multiple, overlapping windows and search pedestrians in each of these windows.

2.1 Sliding Window Based Pedestrian Detection

Recently, researchers have put more effort into sliding window approach as this approach simplifies the problem of PD to binary classification in a given window. Sliding window-based approaches can be further subdivided into rigid and part-based methods. The rigid schemes consider the window holistically to identify a pedestrian. Part-based schemes, such as [35, 36] identify different parts of a pedestrian in a window, and decide the presence of a pedestrian based on the location and confidence (accuracy) of detected parts. Part-based schemes have been shown to perform better compared to rigid schemes as the decision in these methods is based on the aggregate of decisions for different parts and these schemes can handle occlusion better compared to rigid schemes. However, the higher computational complexity of part-based methods makes them infeasible for real-time applications. Rigid schemes utilize a single feature or multiple features to detect an object. We have categorized the schemes depending on the feature and implementation platform.

Single Feature Pedestrian Detection. In [11], the authors use Haar-like features with Support Vector Machines (SVM) to identify objects in a scene. Their method was advanced in [37] for face detection, where the authors obtained an astounding increase in speed by using integral images to compute Haar-like features. Furthermore, cascaded boosted trees were used for classification. The method of [37] was used for PD in [38]. However, using Haar-like features for PD detection did not show much promise until recently [4] due to their low accuracy.

A set of rich and compact features was required to improve PD. Rich features were needed to extract maximum information from a window and compactness was needed to better generalize from training to testing. HOG [4] performed both these tasks by including the complete (or rich) gradient information of a window into compact histograms. They trained Linear Support Vector Machine (LinSVM) framework for classification. Furthermore, they developed and used the INRIA pedestrian dataset, which was the most extensive dataset for PD at that time. Resultantly, their method achieved significant improvement in accuracy of PD compared to the previous schemes.

Since its inception, HOG has influenced most of the modern PD methods. In [39], gradients in local patches, similar to HOG, are used to represent shape descriptors. These shape descriptors are combined to create a single feature which is classified using boosted trees. The method is used for PD as well as detection of other objects. Edgelets, used in [40] and [41], have been used to learn and classify body parts with boosted trees. Other variations include distance transform and template hierarchy [42] to match images with templates, granularity-tunable gradients partition to define spatial and angular uncertainty of line segments [43] and its extension to spatiotemporal domain [44], shape features [45] and finally motion based features [46, 47].

Multiple Feature Pedestrian Detection. To further enhance the PD accuracy of HOG, researchers have complimented HOG with other features. Local Binary Pattern (LBP) is a very simple feature based on magnitude comparison of surrounding pixels, and has typically been used for texture classification [48] and face detection [49]. It has also been used in PD [50]. In [51], the authors present a feature combining both HOG and LBP and use linSVM for classification. They show that this combination improves the PD performance under partial occlusion. In [52], the authors use implicit segmentation and divide the image into separate foreground and background, followed by HOG. HOG, LBP and local ternary patterns were combined in [53] for pedestrian and object detection. Gradients information and HOG, textures (co-occurrence matrices), and color frequency are combined in [54]. Partial least squares are used to reduce the dimensions of the feature and SVM is used for classification. HOG has been combined with Haar-like [55], shapelets [39], color self-similarity and motion [56] features as well. Note that none of these features when used independently from HOG has been able to outperform HOG.

2.2 Real Time Pedestrian Detection

Numerous applications require PD at fast rates. For such applications, it is more suitable to use task-specific (GPU, FPGA, ASIC) rather than general-purpose platforms. A fine grain parallel ASIC implementation of HOG-based PD is presented in [7]. In [15], simplified methods are presented for division and square root operations for use in HOG. However, by employing their methods, the accuracy of PD is severely degraded. A multiprocessor system on chip (SoC) based hardware accelerator for HOG feature extraction is described in [16]. In [17], the authors reuse the features in blocks to construct the HOG features of overlapping regions in detection windows and then use interpolation to efficiently compute the HOG features for each window.

In [18], the authors developed an efficient FPGA implementation of HOG to detect traffic signs. In [19], a real-time PD framework is presented which utilizes an FPGA for feature extraction and a GPU for classification. A deep-pipelined single chip FPGA implementation of PD using binary HOG with decision tree classifiers is discussed in [20]. A heterogeneous system is presented in [22] to optimize the power, speed and accuracy.

From the discussion above, we notice that HOG is integral to most PD algorithms. Efforts have been made in the research community to either improve the accuracy of PD or reduce the hardware complexity of HOG. In our study, we are yet to find an effort which analyzes the effects of reducing hardware costs on accuracy of HOG. For real-time PD with power and area constraints, it is imminent to find the set of parameters of HOG that provide the best compromise in terms of computational complexity and accuracy.

3 Overview of HOG

In this section, we present a brief overview of the HOG algorithm for PD. Although HOG can be used in a part-based PD scheme, we limit our discussion to the rigid HOG as described in the original paper [4]. A block diagram showing functional blocks of the algorithm is shown in Fig. 2.

In HOG, a search window is divided into multiple overlapping blocks which are further divided into cells as shown in Fig. 3, where $(w_F, h_F), (w_W, h_W), (w_B, h_B)$, (w_C, h_C), are the frame, window, block and cell (width, height) respectively. N_{bin} is the

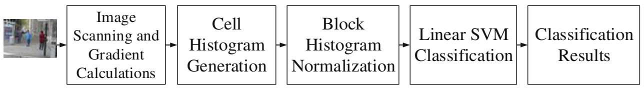


Image Scanning and Gradient Calculations
Cell Histogram Generation
Block Histogram Normalization
Linear SVM Classification
Classification Results

Fig. 2. Block diagram of HOG based pedestrian detection

number of histogram bins. The blocks have an overlap of 50 %, creating a dense grid over the search window. So a single $(w_W \times h_W)$ window has $n_C = w_w/w_C \times h_W/h_C$ cells and $n_B = \left(\frac{w_W - w_C}{w_C} \times \frac{h_W - h_C}{h_C}\right)$ blocks. Gradient features are extracted from these blocks and cells, and are concatenated to create a single feature vector for the whole window.

A filter with coefficients [−1, 0, 1] is applied to the window in horizontal and vertical directions, creating the images G_x and G_y, respectively. These images are used to generate the gradient magnitude image, G_m, and the gradient orientation image, G_o, for each pixel (x, y) as follows.

$$G_m(x,y) = |G(x,y)| = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \tag{1}$$

$$G_o(x,y) = \tan^{-1}\frac{G_y(x,y)}{G_x(x,y)} \tag{2}$$

The histogram used in the feature accumulates the orientation information of an image. Each histogram has multiple bins, where each bin represents a specific orientation in the interval [0, π). The value $G_m(x, y)$ is added to the bin of the histogram which corresponds to $G_o(x, y)$. Such histograms are developed for every cell of the window, as shown in Fig. 3.

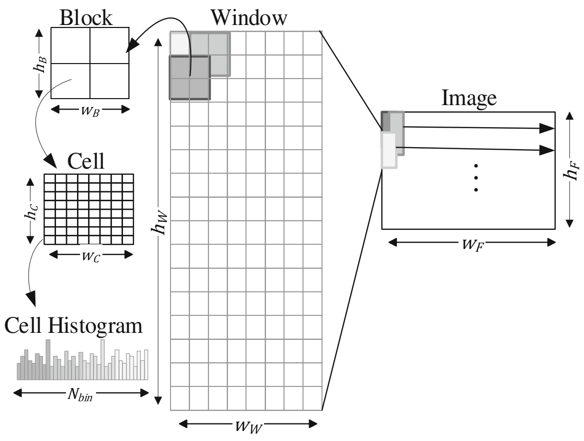


Fig. 3. A depiction of image division for sliding window based object detection. An input image ($w_F \times h_F$) is divided into overlapping windows. The window is divided into overlapping blocks which are further divided into cells. A histogram is generated for every cell.

The cell histograms belonging to a single block are concatenated to form block histogram of length $M = 4 \times N_{bin}$, where, N_{bin} is the number of bins in each cell histogram. Block histograms are further L2-normalized using (3), and then added to the feature vector. *L2-norm* for an un-normalized feature vector *v,* is given by,

$$x_i^b = \frac{v_i^b}{\sqrt{\|v\|_2^2 + \varepsilon^2}}. \tag{3}$$

where $i = 1, \ldots, M, b = 1, \ldots, n_B$, $\|v\|_2^2 = v_1^2 + v_2^2 + \ldots + v_M^2$ and ε is a small constant to avoid division by zero. L2-normalization is performed to improve robustness against illumination changes.

For classification, LinSVM is used. From an implementation perspective, a weight vector is obtained after the training stage. During classification, a dot product of the feature extracted from the window and the weight vector is compared against a threshold. If the dot product is greater than the threshold, then a pedestrian is identified.

4 Hardware Architecture

The hardware implementation of HOG presents a unique challenge, which is quite distinct from the software implementation. First, we cannot store and access a complete frame, and read and write from multiple addresses at once as this will require unrealistically large hardware resources. Second, floating point operations are quite costly in hardware, as they use more FPGA area and runs at a lower frequency; therefore, we avoid them in hardware implementation. Finally, the choice of parameters affects hardware complexity significantly compared to software implementation.

Our key objectives in this implementation are to maintain the maximum accuracy and minimum power consumption while performing real time PD by controlling local features. Hardware/memory optimization is done using optimal values of these features. The optimized architecture thus obtained results in a reduced workload and low bandwidth.

The conceptual block diagram of the proposed HOG Accelerator (HOG-Acc) is shown in Fig. 4. In the following we present a description of the major functional blocks shown in Fig. 4.

4.1 Gradient Computation

To compute the gradient magnitude and orientation, the horizontal and vertical gradient images, i.e., G_x and G_y, need to be generated. Gradient is computed over the 3×3 neighborhood of each pixel; therefore, two line buffers are required to store two consecutive scan lines of the image to maintain a 3×3 neighborhood of every pixel.

A straight forward computation of the gradient magnitude, as given in (1), will require the implementation of the square root operation, which will consume significant hardware resources; thereby, delay and power consumption will increase. In order to

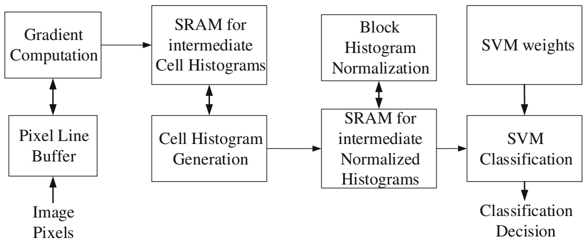

Gradient Computation
SRAM for intermediate Cell Histograms
Block Histogram Normalization
SVM weights
Pixel Line Buffer
Cell Histogram Generation
SRAM for intermediate Normalized Histograms
SVM Classification
Image Pixels
Classification Decision

Fig. 4. Block diagram of hardware architecture. Gradient is computed over input pixels stored in Pixel Line buffer, cell histograms are then built using gradients, SRAM is used to store intermediate cell histograms. Next steps are normalization of histograms generated and finally classification.

reduce the computational complexity, the following approximations from [15] have been used to compute the gradient magnitude and orientation.

$$G_M(x, y) \approx \max((0.875a + 0.5b), a), \tag{4}$$

where,

$$a = \max\big(G_x(x, y), G_y(x, y)\big), \tag{5}$$

and

$$b = \min\big(G_x(x, y), G_y(x, y)\big). \tag{6}$$

The circuitry for gradient magnitude computation is shown in Fig. 5. Equations (5) and (6) are implemented using a single compare operation, while (4) requires four shift operations yielding *0.875a* and *0.5b*, then an adder and one more comparator is used to give the final gradient magnitude.

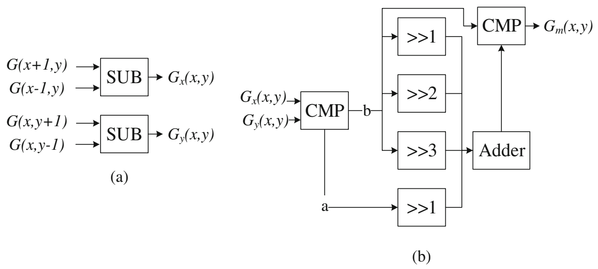


Fig. 5. Gradient magnitude computation module. (a) Simply subtract the horizontal and vertical neighboring pixels to compute the horizontal and vertical gradients. (b) Gradient magnitude is computed by shift and compare operations to implement (4)–(6)

Similarly, a direct implementation of (2) for computing the gradient orientation will require two costly hardware operations: the inverse tangent and division. To reduce the complexity, (2) can be rewritten as

$$G_x(x,y)\tan(G_o(x,y)) = G_y(x,y). \tag{7}$$

The problem of identifying the gradient orientation can be solved using (7) as: multiplying the horizontal gradient value with the values of the right column of Table 1; the product which best matches against the vertical gradient indicates the gradient of the pixel. Note that even the multiplication operation is not required, as the product with the values in the right column of Table 1 can be performed by simple arithmetic shifting.

Table 1. Approximated values of *tanθ*

Tangent	Approximated value
$tan0^\circ$	0
$tan10^\circ$	$2^{-3}+2^{-4}$
$tan20^\circ$	$2^{-2}+2^{-3}$
$tan30^\circ$	$2^{-1}+2^{-4}+2^{-6}$
$tan40^\circ$	$2^{-1}+2^{-2}+2^{-4}$

The circuit to compute the histogram bin is shown in Fig. 6. It consists of two parts, one deals with the quadrant decision and other decides the bin. Comparing horizontal and vertical neighboring pixels sets the Q-flag value which indicates whether the bin lies in first or second quadrant. Once we know the quadrant, we have to decide which histogram bin the orientation value lies in. By using Table 1 to approximate the value of tangent function at different orientations, complex operations such as inverse tangent and division can be avoided. The hardware utilizes only comparators, shifters and adders, hence reducing the complexity significantly.

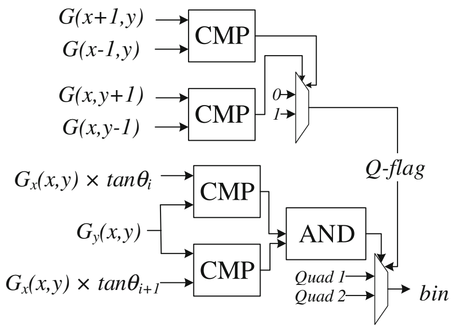


Fig. 6. Bin computation module: bin quadrant is decided using a Q-flag, which is computed by comparing horizontal and vertical pixels, histogram bin is then decided implementing (7) using comparators and AND gate.

It has been shown in [8] that bit-width assigned to magnitude has a significant impact on accuracy, throughput and power consumption as it affects the data sizes at all the next stages. In [8] fixed-point implementation is considered and bit width of gradient magnitude is optimized as 13 bits (9:4 (integer: fractional)). We argue that using only integer values of gradient magnitude can further improve the accuracy, throughput and power consumption. The details are given in Sect. 4. The key insight is that by using integer values for gradient magnitudes, we can remove the histogram values which are less significant. The advantages are twofold. (1) It reduces the hardware complexity due to reduced bit width and integer operations. (2) It improves the accuracy because removing these petty gradient magnitudes enhances the feature vector for training and classification.

4.2 Cell Histogram Generation

We propose a parallel Cell Histogram Generation (CHG) module as shown in Fig. 7. Gradient magnitudes and orientation bins for every $w_C \times h_C$ pixels are given as input to CHG. Decoders and adders are used to build the histogram. Each bin value is given as input to the decoder. Only one output is set to '1' corresponding to the specific bin; gradient magnitude for that bin hence propagates to the input of adder, where all the magnitudes of the same bin are added.

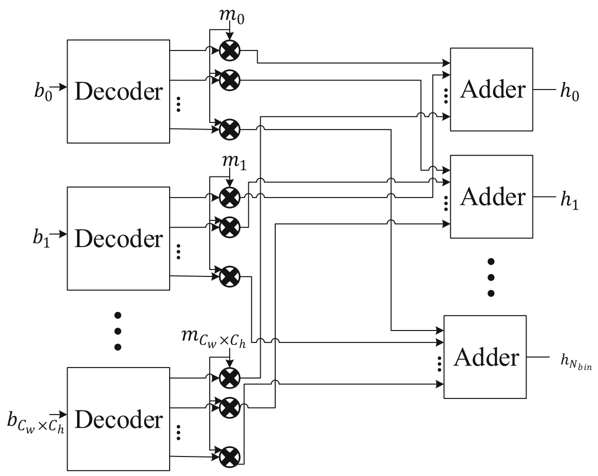


Fig. 7. Cell Histogram Generation (CHG) Engine: Histogram bins and gradient magnitudes are given as input and cell histograms are generated.

The decoder size is dependent on the number of bits required to represent single bin, i.e. if number of histogram bins increase the size of decoder increases. On the other hand, the cell size $(C_{size} = w_C \times h_C)$ affects the number of decoders as the total number of decoders required equals C_{size}. Multipliers required for CHG are dependent on both

N_{bin} and C_{size}. Multipliers in each stage depend on N_{bin} while number of stages depends on C_{size}. Finally, the number of adders is equivalent to the N_{bin} chosen. Adder size, however, varies according to C_{size}. We can clearly see that the complexity of CHG is strongly dependent on N_{bin} and C_{size}.

Since pixels are coming row by row, we have to maintain cell histograms for multiple blocks and windows as each row has multiple windows. Therefore, the gradient magnitudes and orientations computed for every w_C pixels (one cell) are concatenated and stored in memory. Pixels of row index which is a multiple of h_C indicate the completeness of cell. This row is directly stored into registers. At the start of every such row, respective values of previous rows for the particular cell are read into registers from block RAM every clock cycle. As we have considered $w_C = h_C$ the cell completes in horizontal and vertical directions simultaneously. Hence, the number of shift registers required is equivalent to h_C. Each shift register stores magnitudes and bins for w_C pixels. After w_C cycles the data of one cell is completed so it is shifted to the memory, which in turn writes the data for the previous row in the next register.

The resultant cell histogram is given to the next stage for processing. This is done every time the new cell is completed. i.e. when the row index is a multiple of h_C and column index is a multiple of w_C. The cell histograms for multiple windows in a frame are stored in memory while they are shifted to registers for each active window (the window whose cells histograms are completed).

4.3 Block Histogram Normalization

Cell histograms are maintained in the memory till four neighboring cells are completed and a block is obtained. Note that the memory required to store the cell histograms increases with smaller cells (more cells per row and column) and larger number of histogram bins for every cell (more data per cell). In other words, C_{size} affects the memory locations required while N_{bin} influences the width of each location.

The histogram is normalized using the Block Histogram Normalization Engine (BHN) shown in Fig. 8. Normalization is performed every time a new block is completed. Each histogram value in a block is squared and added. The sum is given as input to inverse square root module which is approximated using "fast inverse square root" algorithm [30]. In summary, logical shifting, subtraction and finally one iteration of Newton's method approximates the inverse square root. Finally, the result of inverse square root is multiplied with each histogram value to generate the normalized block histogram.

It is seen that the number of multipliers in BHN depends on the size of the block histogram, which is related to number of bins assigned to each cell histogram. Adding a single bin to cell histogram adds eight multipliers to the hardware. The adder size also increases proportionately.

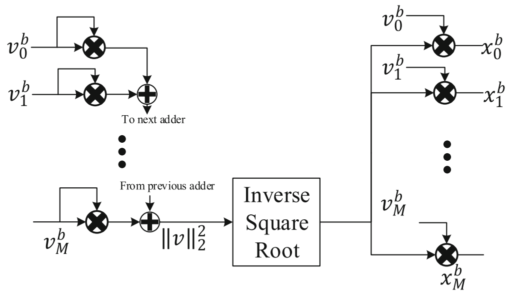


Fig. 8. Block Histogram Normalization (BHN) engine: un-normalized Block histograms (concatenated cell histograms) are used as inputs to generate normalized block histograms.

4.4 SVM Classification

The normalized histograms obtained from the BHN block are again stored in the memory. Once normalized histograms for the whole window are available classification can be performed which can consume a fair amount of memory. Performing classification for the whole window at once also requires a large number of multipliers and adders. The situation gets worse as the feature vector size increases with smaller cell sizes or large number of bins. Therefore, we have opted for partial classification by dividing the classification for the whole window into multiple stages. The hardware shown in Fig. 9. is reused at every stage. The strategy behind reusing the hardware is very straightforward. Since it takes w_C cycles to completely process a cell, we have reused the same hardware over these w_C cycles doing partial classification every N_B/w_C blocks. So the number of partial classification stages is equal to C_w. The results of each stage are accumulated to get the final classification result.

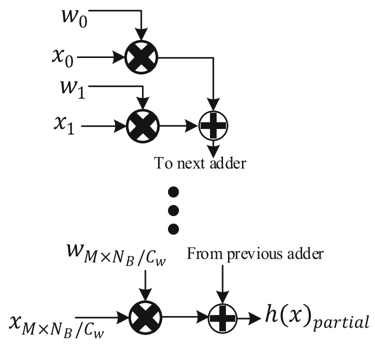


Fig. 9. Partial Classification Engine (PCE), single stage of LinSVM classification to be performed for whole window. Inputs are normalized histograms while output is the partial classification result.

The key observation is that the cell size effects the hardware complexity in two ways. First, it has a direct impact on feature vector size. Second, larger the cell size, more cycles will be available to perform classification, thereby, smaller hardware is required for partial classification.

5 Results and Discussion

In this section, we evaluate our hardware implementation for multiple cell sizes and histogram bins to obtain optimal set of these parameters. Results are presented for full-HD (1920 × 1080) resolution videos. Window size is considered to be 64 × 128. Block size is 2 × 2 cells, while block and window step size is one cell for both horizontal and vertical directions. Scale factor to rescale images is set to 1.05. This results in 45 scales to be processed per frame. Other parameters depend on the choice of cell size and histogram bins.

Here, we first present our experimental setup then we analyze the effect of different cell sizes and histogram bins on accuracy, throughput and power. Using these results, parameters yielding least power and maximum throughput with negligible loss in accuracy are selected. Finally, using these parameters comparison with the state of the art object detection implementations is presented.

5.1 Experimental Setup

We have implemented our system on Xilinx Virtex 7 (XC7VX485T) FPGA. There are 75,900 slices, 607,200 Configurable Logic Blocks (CLBs) and 485,760 logic cells in this FPGA. Moreover, 37,080 Kb block RAM and 2,800 DSP slices are present. Image rescaling and window sampling is done for positive and negative images and then sent to HOG-Acc for processing which returns the detection result. Processing 45 scales requires a large amount of memory and pipelined stages so we have utilized the time multiplexing approach of [21]. The host software is written using Visual Studio 2012 and Verilog is used for HOG-Acc design. Design is synthesized using Xilinx ISE 14.7 and along with Modelsim 10.2, a hardware/software co-simulation is performed to verify the implementation functionality.

5.2 Accuracy Analysis

We have used INRIA dataset [31], to evaluate our HOG implementation. There are several other datasets available for pedestrian detection evaluation like Caltech [32], ETH [33], and Daimler [3]. We have, however, restricted our results to INRIA because it provides us with a reasonable variety of images with different poses and backgrounds so these results can be generalized to other datasets and real life scenarios.

All detection results are collected, and afterwards recall is calculated from number of true positives (TP) and false negatives (FN) as shown in (8).

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

A false positive per window (FPPW) of *10*$^{-4}$ is mostly considered in literature for pedestrian detection results. We also present the *Miss Rate (1-Recall)* results for *FPPW* = *10*$^{-4}$ for multiple cell sizes and number of bins. The results are shown in Fig. 10. It is seen that generally larger histogram bins gives better detection rates. This is obvious, as more histogram bins allow fine division gradient orientations, hence better feature vector for training and classification. On the other hand, improvement is seen in detection rates by increasing cell size up to a certain value and it drops increasing cell sizes too much. Smaller cell sizes provide a dense grid of blocks and windows in a frame, therefore, using smaller cells would improve accuracy. However, using too small cell sizes results in degraded performance because there are not enough distinguishing features within the cells. Minimum miss rate of 12 % is achieved at $(C_{size}, N_{bin}) = (7 \times 7, 11)$.

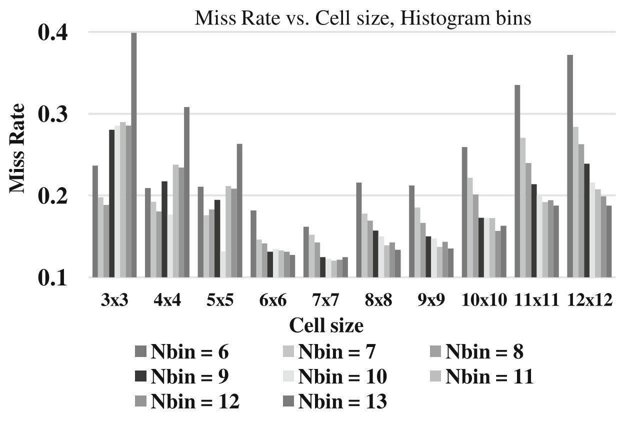
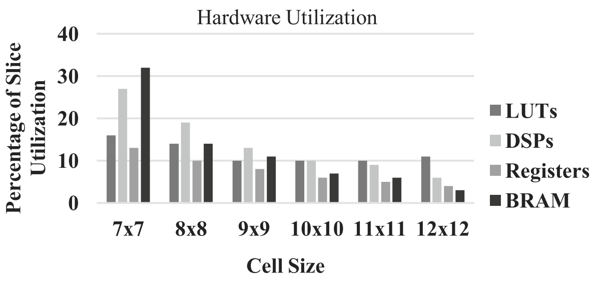


Fig. 10. Accuracy analysis, miss rate generally reduces with increasing cell sizes and decreasing number of bins

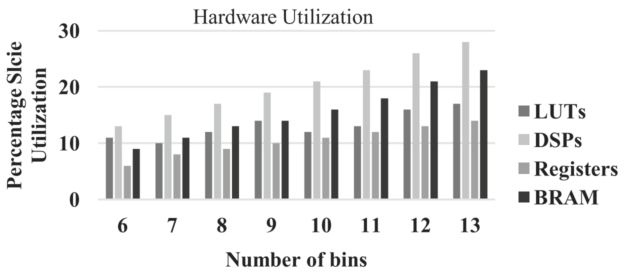
5.3 Throughput and Power Consumption Analysis

Power consumption and throughput are directly related to the hardware resources used. In the previous section it is seen that the cell size and histogram bins has significant impact on hardware complexity. The effect on different hardware components for different cell sizes and number of bins for a single core is shown in Fig. 11. We see a significant reduction in hardware resources by increasing cell size or reducing number of bins. The reasons being discussed in previous section for independent blocks.

Number of frames processed per second (fps) is dependent on the maximum frequency at which the hardware can operate. In our hardware architecture it is mainly dependent on the size of partial classification engine and the block normalization engine. As discussed in previous section, the complexity of PCE is heavily dependent



(a) Percentage of slice utilization for multiple cell sizes and fixed N_{bin}=9.



(b) Percentage of slice utilization for multiple histogram bins and fixed C_{size}=8x8

Fig. 11. Hardware utilization comparison. Breakdown of usage of multiple slices of Xilinx FPGA with varying cell size and histogram bins. Increasing any one of them results in increased hardware complexity.

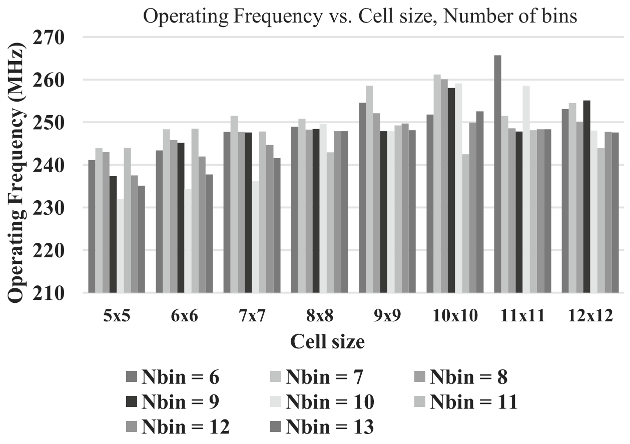


Fig. 12. Throughput analysis, an increase in throughput is seen for bigger cell sizes and histogram bins.

on C_{size} while that of BHN depends on both C_{size} and N_{bin}. Figure 12 shows the results. We get the maximum frequency at the point where both PCE and BHN have overall minimal hardware complexity. Specifically, the $(C_{size}, N_{bin}) = (11 \times 11, 6)$.

We have used Xilinx Xpower analyzer (XPA) 14.7 to estimate the deviations in power consumption by varying the parameters. We have simulated the hardware and created 'Value Change Dump' (vcd) files are used to set the toggle rates of all signals. Post place and route results are obtained and are shown in Fig. 13. Power consumption increases by reducing the cell size or increasing the histogram bins. This is fairly understandable due to the fact that both these parameters increase the hardware complexity due to increase in the feature vector size. Minimum power consumption is 9.98 W with $(C_{size}, N_{bin}) = (12 \times 12, 6)$, the maximum cell size and minimum histogram bins as expected.

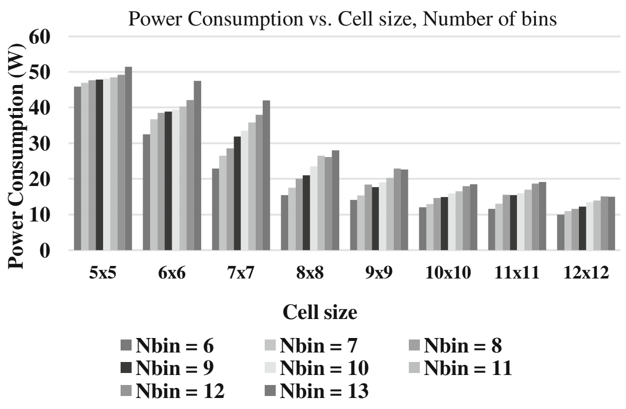


Fig. 13. Power consumption analysis, large cell size and smaller number of histogram bins results in low power consumption.

5.4 Choice of Parameters

We have seen from the previous analysis that there does not exist a set of parameters which give us best accuracy, power and throughput. Improving the accuracy worsens the power and throughput while maintaining minimum power and maximum throughput severely degrades the accuracy. Similarly, trying to improve throughput may degrade power consumption significantly and vice versa. However, accuracy is changing very slightly at certain regions in Fig. 9. Similarly, there are more than one sets of parameters which give almost the same power consumption. This allows us to select the best of one of these metrics while slightly compromising on another metric. We can achieve best results by selecting $(C_{size}, N_{bin}) = (9 \times 9, 10)$. Further we obtained results for miss rate by changing bit-width for this optimal parameter set. The results are shown in Fig. 14. Bit-width is hence set to eight bits as it gives maximum accuracy and minimum hardware complexity. Note that this further results in reduced bit-width in all the next blocks.

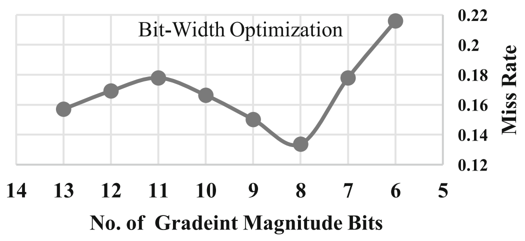


Fig. 14. Variation in miss rate for $(C_{size}, N_{bin}) = (9 \times 9, 10)$.

Parameters optimized for low power and high speed are shown in Tables 2 and 3 comparing the throughput and energy consumption results with the other state of the art FPGA and GPU implementations. We have presented three results. (1) HOG_{CONV}, which shows the results for conventionally used parameters. We can achieve 32 fps for full-HD while dissipating 0.656 J/frame and 15 % miss rate. (2) HOG_{OCB}, presents results for optimized cell size and histogram bins. Frame rate achieved by optimizing cell size and histogram bins is 39.2 fps with energy consumption of 0.484 J/frame while maintaining a miss rate at 15 %. Gradient magnitude bit-width is considered to be 13 bits. (3) Finally, $HOG_{OCB\text{-}RF}$, in which features are further refined by removing insignificant gradients, is presented. This results in a frame rate of 42.7 fps while energy consumption is 0.45 J/frame at 13 % miss rate.

Table 2. Comparison of parameters and throughput for various GPU and FPGA implementations

	Cell size	Histogram bins	Win. stride	# scales	Resolution	Windows/frame	FPS
GPU implementation							
[22, 24]	8 × 8	9	8	37	1024 × 768	–	17
[25]	8 × 8	–	–	>1	640 × 480	4,096	57
[26]	8 × 8	8	2	>1	640 × 480	50,000	23.8
[27]	8 × 8	9	8	1	640 × 480	–	32
[28]	8 × 8	9	4	>1	1280 × 960	150,000	2.4
[29]	8 × 8	9	–	>1	640 × 480	–	5.6
FPGA implementation							
[18]	8 × 8	8	4	>1	320 × 240	3,615	38
[19]	8 × 8	9	–	1	800 × 600	1000	>10
[20]	8 × 8	8	9	1	640 × 480	1,540	62.5
[21]	8 × 8	9	8	18	1920 × 1080	27,960	64 (estimated)
[22]	8 × 8	9	8	13	1024 × 768	20,868	13
[23]	8 × 8	8	4	>1	640 × 480	56,466	30 (estimated)
[6]	8 × 8	9	8	1	800 × 600	5,580	72
[8]	8 × 8	9	4	34	640 × 480	121,210	68.18
[8]	8 × 8	9	4	34	1600 × 1200	1,049,886	10.41 (estimated)
HOG_{CONV}	8 × 8	9	8	45	1920 × 1080	264,062	32
HOG_{OCB}	9 × 9	10	8	45	1920 × 1080	264,062	39.2
$HOG_{OCB\text{-}RF}$	9 × 9	10	8	45	1920 × 1080	264,062	42.7

Table 3. Comparison of parameters and energy consumption for various GPU and FPGA implementations

	Cell size	Histogram bins	Win. stride	# scales	Resolution	Windows/frame	Power (W)	Energy (J/Frame)
[8] (GPU)	8 × 8	9	4	34	640 × 480	121,210	225	17
[8] (FPGA)	8 × 8	9	4	34	640 × 480	121,210	37	0.54
HOG_{CONV}	8 × 8	9	8	45	1920 × 1080	264,062	21	0.656
HOG_{OCB}	9 × 9	10	8	45	1920 × 1080	264,062	19	0.485
$HOG_{OCB\text{-}RF}$	9 × 9	10	8	45	1920 × 1080	264,062	19.276	0.451

6 Conclusion

We have presented fully parallel architectures for various modules of pedestrian detection system utilizing Histogram of oriented gradients (HOG). HOG has shown high detection accuracy but the detection speed and power consumption are major bottlenecks for real time embedded applications. We have optimized parameters, cell size and histogram bins, to achieve low power and high throughput while maintaining the detection accuracy. Feature refinement is done to further improve the results.

Combination of optimal parameters and our hardware accelerator results in a frame rate of 42.7 fps for full-HD resolution and lowers the energy consumption by 97.34 % and 16.4 % while improving the accuracy by 2 % as compared to state of the art GPU and FPGA implementations respectively. This work can be extended to use multiple cores on a single FPGA or using multiple FPGAs to further increase throughput while an ASIC implementation would greatly reduce the power consumption. It can also be extended to include other features and classifiers or combinations of those to optimize for objects other than pedestrians.

Acknowledgments. This work was supported by the Center of Integrated Smart Sensors funded by Ministry of Science, ICT & Future Planning as Global Frontier Project (CISS-2013M3 A6A6073718).

References

1. Shankar, U.: Pedestrian roadway fatalities. Department of Transportation Technical report (2003)
2. Geronimo, D., Lopez, A.M., Sappa, A.D., Graf, T.: Survey on pedestrian detection for advanced driver assistance systems. IEEE Trans. Pattern Anal. Mach. Intell. **32**(7), 1239–1258 (2010). 1, 2, 10, 16, 18
3. Ess, A., Leibe, B., Schindler, K., Van Gool, L.: A mobile vision system for robust multi-person tracking. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–8 (2008)
4. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: Proceedings of IEEE Conference on Computer Vision Pattern Recognition, vol. 1, pp. 886–893, June 2005
5. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. IJCV **60**(2), 91–110 (2004)

6. Mizuno, K., Terachi, Y., Takagi, K.: Architectural study of HOG feature extraction processor for real-time object detection. In: Proceedings of IEEE Workshop Signal Processing Systems, pp. 197–202, October 2012
7. Takagi, K., et al.: A sub-100-milliwatt dual-core HOG accelerator VLSI for real-time multiple object detection. In: ICASSP (2013)
8. Ma, X., Najjar, W.A., Roy-Chowdhury, A.K.: Evaluation and acceleration of high-throughput fixed-point object detection on FPGAs. IEEE Trans. Circ. Syst. Video Technol. **25**(6), 1051–1062 (2015)
9. Dollar, P., Wojek, C., Schiele, B., Perona, P.: Pedestrian detection: an evaluation of the state of the art. IEEE Trans. Pattern Anal. Mach. Intell. **34**(4), 743–761 (2012)
10. Khan, A., Khan, M.U.K., Bilal, M., Kyung, C.-M.: Hardware architecture and optimization of sliding window based pedestrian detection on FPGA for high resolution images by varying local features. In: VLSI-SoC 2015, pp. 142–148 (2015)
11. Papageorgiou, C., Poggio, T.: A trainable system for object detection. Int. J. Comput. Vis. **38**(1), 15–33 (2000)
12. Oren, M., Papageorgiou, C., Sinha, P., Osuna, E., Poggio, T.: Pedestrian detection using wavelet templates. In: Proceedings of IEEE Conference on Computer Vision Pattern Recognition, pp. 193–199, June 1997
13. Lowe, G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60** (2), 91–110 (2004)
14. Cheng, H., Zheng, N., Qin, J.: Pedestrian detection using sparse Gabor filter and support vector machine. In: Proceedings of IEEE Intelligent Vehicles Symposium, pp. 583–587, June 2005
15. Chen, P.Y., Huang, C.C., Lien, C.Y., Tsai, Y.H.: An efficient hardware implementation of HOG feature extraction for human detection. IEEE Trans. Intell. Transp. Syst. **15**(2), 656–662 (2014)
16. Lee, S.E., Min, K., Suh, T.: Accelerating histograms of oriented gradients descriptor extraction for pedestrian recognition. Comput. Elect. Eng. **39**(4), 1043–1048 (2013)
17. Pang, Y., Yuan, Y., Li, X., Pan, J.: Efficient HOG human detection. Signal Process. **91**(4), 773–781 (2011)
18. Hiromoto, M., Miyamoto, R.: Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features. In: Proceedings of IEEE ICCVW, pp. 894–899 (2009)
19. Bauer, S., Kohler, S., Doll, K., Brunsmann, U.: FPGA-GPU architecture for kernel SVM pedestrian detection. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 61–68, June 2010
20. Negi, K., Dohi, K., Shibata, Y., Oguri, K.: Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm. In: Proceedings of International Conference on FPT, pp. 1–8, 12–14 December 2011
21. Hahnle, M., Saxen, F., Hisung, M., Brunsmann, U., Doll, K.: FPGA based real-time pedestrian detection on high-resolution images. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 629–635, June 2013
22. Blair, C., Robertson, N., Hume, D.: Characterizing a heterogeneous system for person detection in video using histograms of oriented gradients: power versus speed versus accuracy. IEEE J. Emerg. Sel. Top. Circ. Syst. **3**(2), 236–247 (2013)
23. Kadota, R., Sugano, H., Hiromoto, M., Ochi, H., Miyamoto, R., Nakamura, Y.: Hardware architecture for HOG feature extraction. In: 5th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP), pp. 1330–1333 (2009)
24. OpenCV: http://opencv.org/

25. Sudowe, P., Leibe, B.: Efficient use of geometric constraints for sliding-window object detection in video. In: Crowley, J.L., Draper, B.A., Thonnat, M. (eds.) ICVS 2011. LNCS, vol. 6962, pp. 11–20. Springer, Heidelberg (2011)
26. Machida, T., Naito, T.: GPU & CPU cooperative accelerated pedestrian and vehicle detection. In: IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pp. 506–513 (2011)
27. Yan-ping, C., Shao-zi, L., Xian-ming, L.: Fast HOG feature computation based on CUDA. IEEE Int. Conf. Comput. Sci. Autom. Eng. (CSAE) **4**, 748–751 (2011)
28. Bilgic, B., Horn, B.K.P., Masaki, I.: Fast human detection with cascaded ensembles on the GPU. In: 2010 IEEE Intelligent Vehicles Symposium (IV), pp. 325–332 (2010)
29. Prisacariu, V., Reid, I.: fastHOG - a real-time GPU implementation of HOG. Department of Engineering Science, Oxford University, Technical report 2310/09 (2009)
30. http://en.wikipedia.org/wiki/Fast_inverse_square_root
31. INRIA Person Dataset. http://pascal.inrialpes.fr/data/human/
32. Dollar, P., Wojek, C., Schiele, B., Perona, P.: Pedestrian detection: a benchmark. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 304–311 (2009)
33. Enzweiler, M., Gavrila, D.: Monocular pedestrian detection: survey and experiments. IEEE Trans. Pattern Anal. Mach. Intell. **31**(12), 2179–2195 (2009)
34. Gu, C., Lim, J.J., Arbelaez, P., Malik, J.: Recognition using regions. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)
35. Felzenszwalb, P., McAllester, D., Ramanan, D.: A discriminatively trained, multiscale, deformable part model. In: IEEE Conference on Computer Vision and Pattern Recognition (2008)
36. Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part based models. IEEE Trans. Pattern Anal. Mach. Intell. **99** (2009). PrePrints
37. Porikli, F.M.: Integral histogram: a fast way to extract histograms in Cartesian spaces. In: IEEE Conference on Computer Vision and Pattern Recognition (2005)
38. Viola, P.A., Jones, M.J., Snow, D.: Detecting pedestrians using patterns of motion and appearance. Int. J. Comput. Vis. **63**(2), 153–161 (2005)
39. Sabzmeydani, P., Mori, G.: Detecting pedestrians by learning shapelet features. In: IEEE Conference on Computer Vision and Pattern Recognition (2007)
40. Wu, B., Nevatia, R.: Detection of multiple, partially occluded humans in a single image by Bayesian combination of edgelet part det. IEEE Int. Conf. Comput. Vis. **6**(10), 11 (2005)
41. Wu, B., Nevatia, R.: Cluster boosted tree classifier for multi-view, multi-pose object detection. In: ICCV (2007)
42. Gavrila, D.M.: A Bayesian, exemplar-based approach to hierarchical shape matching. IEEE Trans. Pattern Anal. Mach. Intell. **29**(8), 1408–1421 (2007)
43. Liu, Y., Shan, S., Zhang, W., Chen, X., Gao, W.: Granularity tunable gradients partition descriptors for human detection. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)
44. Liu, Y., Shan, S., Chen, X., Heikkila, J., Gao, W., Pietikainen, M.: Spatial-temporal granularity-tunable gradients partition (STGGP) descriptors for human detection. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part I. LNCS, vol. 6311, pp. 327–340. Springer, Heidelberg (2010)
45. Gavrila, D.M., Philomin, V.: Real-time object detection for smart vehicles. In: IEEE International Conference on Computer Vision, pp. 87–93 (1999)
46. Dalal, N., Triggs, B., Schmid, C.: Human detection using oriented histograms of flow and appearance. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3952, pp. 428–441. Springer, Heidelberg (2006)

47. Wojek, C., Walk, S., Schiele, B.: Multi-cue onboard pedestrian detection. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)
48. Ojala, T., Pietikainen, M., Maenpaa, T.: Multiresolution grayscale and rotation invariant texture classification with local binary patterns. IEEE Trans. Pattern Anal. Mach. Intell. **24** (7), 971–987 (2002)
49. Rodriguez, Y.: Face detection and verification using local binary patterns. Ph.D. thesis, EPF Lausanne (2006)
50. Zheng, Y., Shen, C., Hartley, R.I., Huang, X.: Effective pedestrian detection using center-symmetric local binary/trinary patterns. In: CoRR (2010)
51. Wang, X., Han, T.X., Yan, S.: An HOG-LBP human detector with partial occlusion handling. In: IEEE International Conference on Computer Vision (2009)
52. Ott, P., Everingham, M.: Implicit color segmentation features for pedestrian and object detection. In: IEEE International Conference on Computer Vision (2009)
53. Hussain, S., Triggs, B.: Feature sets and dimensionality reduction for visual object detection. In: British Machine Vision Conference (2010)
54. Schwartz, W., Kembhavi, A., Harwood, D., Davis, L.: Human detection using partial least squares analysis. In: IEEE International Conference on Computer Vision (2009)
55. Wojek, C., Schiele, B.: A performance evaluation of single and multi-feature people detection. In: Rigoll, G. (ed.) DAGM 2008. LNCS, vol. 5096, pp. 82–91. Springer, Heidelberg (2008)
56. Walk, S., Majer, N., Schindler, K., Schiele, B.: New features and insights for pedestrian detection. In: IEEE Conference on Computer Vision and Pattern Recognition (2010)