Article

Adaptive Real-Time Object Detection for Autonomous Driving Systems

Maryam Hemmati [1,*], Morteza Biglari-Abhari [1] and Smail Niar [2]

[1] Department of Electrical, Computer, and Software Engineering, The University of Auckland; m.hemmati@auckland.ac.nz (M.H.) ; m.abhari@auckland.ac.nz (M.B.-A.)
[2] Institut National des Sciences Appliquées (INSA) Hauts-de-France, Université Polytechnique Hauts-de-France; smail.niar@uphf.fr (S.N.)
* Correspondence: m.hemmati@auckland.ac.nz

Abstract: Accurate and reliable object detection is one of the main tasks of Autonomous Driving Systems (ADS). Considering various circumstances for different types of objects and obstacles on the road results in more intensive computations and more complicated systems. Moreover, the stringent real-time requirements of ADS, resource constraints, and energy efficiency considerations on embedded platforms add to the design complications. This work presents dedicated hardware accelerators for pedestrian detection and vehicle detection in different environmental conditions, which are implemented on FPGA. These lighting conditions include three scenarios of day, dusk, and dark. We take a hardware-software co-design approach on Zynq UltraScale+ MPSoC and develop a dynamically reconfigurable ADS that adapts itself to the environment lighting conditions. Our analysis of the results shows that the system adaptability is achieved with minimal resource overhead, while more reliability and robustness are added to the ADS.

Keywords: Real-time detection; hardware accelerator; partial reconfiguration; pedestrian detection; vehicle detection; FPGA; Adaptive ADS; HOG; SVM; DBN

Citation: Hemmati, M.; Biglari-Abhari, M.; Niar, S. Title. *J. Imaging* **2022**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

1. Introduction

Autonomous driving systems (ADS) will be used more widely when their use on the roads is legislated. ADS legal implications are mostly related to their reliability and safety concerns, which could be addressed when these systems guarantee reliable actions in the case of any hazardous situation. It is expected that ADS provide safer and more reliable driving than human drivers. Accurate obstacle detection is an essential prerequisite to make a reliable decision in these autonomous systems. There are various types of obstacles, and they could appear on the road in different environmental conditions. Change of environment could affect the appearance of objects and make the detection task more challenging. All these variations should be taken into account for robust detection in a system which is designed for a safety-critical application such as ADS. Adaptive designs where the system changes its functionality based on the environmental conditions could be a useful solution in these scenarios.

Moreover, the safety-critical nature of ADS imposes hard real-time requirements to the system. Real-time systems are responsible for completing their task within a specified period, otherwise, the task is considered as failed. Hard real-time systems, also known as immediate real-time systems, should be able to complete their operation within the stringent deadline. While missing a deadline in soft real-time systems may lead to a significant loss, it would be catastrophic in hard real-time systems. The simple example of such hard real-time applications is the anti-lock brakes in a car. Time constraints associated with hard real-time applications add more complexity to the design of these systems.

We present an adaptive ADS capable of detecting pedestrians and vehicles in different environmental conditions. Implemented on Zynq UltraScale+ MPSoC, a specific memory hierarchy [1] is employed in our hardware implementations to maximize the data reuse and detection throughput. Hardware implementations of the vehicle detection algorithms are discussed, and the adaptability of the system for vehicle detection is achieved through dynamic and partial reconfiguration of FGPA fabric on programmable logic (PL). Through partial reconfiguration, the system maintains its functionality of detecting pedestrians, while it changes its vehicle detection algorithm on the fly. The reconfiguration process is initiated and triggered by the processing system (PS) on Zynq UltraScale+ MPSoC. Our approach in partial reconfiguration of the FPGA results in added adaptability to the system with negligible increased hardware resources.

The rest of the paper is organized as follows. A review of object detection methods is provided in Section 2. Section 3 presents our pedestrian detection hardware accelerator for single and multi-scale detection. Vehicle detection methods during different lighting conditions and their corresponding hardware accelerators are explained in Section 4. The evaluation results including hardware implementations on PL and algorithm profiling on the PS are discussed in Section 5 followed by a discussion on the advantages of partial and dynamic reconfiguration in providing the adaptability of system with minimal resource overhead. Concluding remarks are discussed in Section 6.

2. Literature Review and Background

During the last few decades, various detection algorithms have been proposed and evaluated for object detection. These methods could be divided into different categories of shape-based detection, motion-based detection, and a combination of shape and motion-based detection. They range from conventional machine learning (ML) approaches [2–6] to deep learning (DL) based techniques [7–9]. In conventional ML approaches, the features are extracted and described through some human-defined algorithms and are passed through a classification stage for the final decision. However, in the DL approach, both feature extraction and classification stages are managed within the network, and it is only the art of designing suitable network architectures so that the required features could be extracted and distinguished efficiently.

Several algorithms are developed in the context of conventional ML. However, most of them rely on a few well-known feature descriptors with some modification in either extraction algorithm or classification structure. One of these early developed features is Haar-like features where a wavelet template is used to define the shape of an object in terms of a subset of the wavelet coefficients of the image [10]. Cascades of Haar-like features proposed by Viola and Jones in 2001 [2] is one of the other earliest methods used for object detection. Originally developed for face detection, this method has the advantage of low computation by introducing a new image representation called "integral image" and yields the detection rate of 15 frame per second (fps) in the original work of face detection [2]. A modified version of AdaBoost classifier is used for the classification purpose in this study [3]. This approach is tailored to human detection by taking into account the motion information by Viola and Jones in 2003 [4].

Histogram of oriented gradients (HOG) introduced by Dalal and Triggs in 2005 is the other well known early approach in human detection [5]. HOG features are considered as one of the most efficient and promising features for human detection within the context of conventional machine learning approaches where the features are handcrafted. These features are usually used in conjunction with a classifier such as AdaBoost [3] or support vector machine (SVM) [6] to detect specific class of objects [11]. HOG features have also been employed in the detection of other object classes such as vehicles and have shown reasonably accurate detection results compared to the other vehicle detection algorithms widely used [12].

Several research works have studied detection of the car either based on its appearance, its motion, or a combination of both [12]. While motion-based methods look at the sequence of frames and employ both detection and tracking algorithms [13,14], appearance-based detection mainly relies on the pixel information of one image frame. In general, these methods extract vehicle appearance features and compare them with a pre-trained model through the classification stage. Various visual features of vehicles are used for this purpose, including the overall shape of car, edges, corners, underneath shadow of the vehicle, headlight or taillight position, and their color. However, all of these features are somehow affected by the environmental condition, which makes the challenge of accurate detection even more complicated.

Modern approaches for object detection are mostly based on deep neural networks (DNN) and convolutional neural networks (CNN). In these approaches, the features are not extracted by a human. Instead, the first few layers of the networks are meant to extract the features and build up more complicated features through the network layers. Final layers of the network act as the classification stage to make the final decision and classify the image into several categories. One of the main concerns in DL-based detection is the high computation requirements which need to be addressed properly. These concerns are partly addressed by employing region-based convolutional network (R-CNN), fast region-based convolutional network (Fast R-CNN), and some other lightweight variations of CNN [7–9].

3. Pedestrian Detection

Pedestrian detection is one of the vital tasks in driver assistance systems (DAS) and autonomous driving systems (ADS) with the aim of bringing more safety to our daily transportation. It is considered as one of the most challenging tasks due to the variation of human poses and their appearance. Reliable detection of pedestrians and humans is also important in several other domains such as video surveillance, and robotics.

3.1. Detection Algorithm

The DL approaches in object detection achieve more accurate results at the cost of more computation. Although in some scenarios, the traditional ML approaches might achieve lower detection accuracy compared to DL-based techniques, these methods are considered as a good candidate in real-time applications with constrained computation power. The combination of HOG feature extractor and linear SVM has shown its competency in human detection, and we base our pedestrian detection on this method.

The principle of object detection in the traditional approach includes two different stages of feature extraction and object classification. At the first stage, specific features of an image are extracted. In the next stage, the classifier decides whether the object belongs to the particular class based on the calculated features in the initial stage. Figure 1 shows a block diagram of object detection using HOG feature extractor followed by SVM classifier. In this method, the HOG features of the input image are first calculated and then passed to the SVM classifier. The classifier compares the HOG features of input with its pre-trained model data, which represents the human model in this case. The result of classifier defines if the image belongs to the pedestrian class or not.

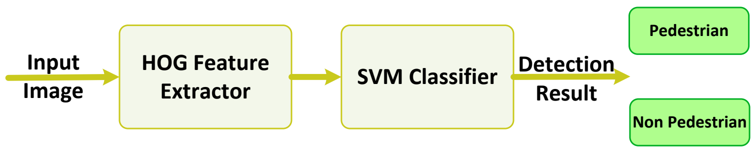

Figure 1. Block diagram of detection based on HOG feature extractor and SVM classifier.

Calculating HOG features incorporates dividing the input image into small parts called cells, normally 8x8 pixels as shown in Figure 2. Then the gradients in both x and y direction are calculated for each pixel within the cell. Simple $[-1,0,1]$ and $[-1,0,1]^T$ gradient filters are applied to the pixel value $f(x,y)$ in order to obtain both $f_x(x,y)$ and $f_y(x,y)$ which are defined as

$$f_x(x,y) = f(x+1,y) - f(x-1,y) \tag{1}$$

$$f_y(x,y) = f(x,y+1) - f(x,y-1) \tag{2}$$

The gradient magnitude $m(x,y)$ and gradient direction $\theta(x,y)$ are then computed as

$$m(x,y) = \sqrt{f_x^2(x,y) + f_y^2(x,y)} \tag{3}$$

$$\theta(x,y) = \arctan \frac{f_y(x,y)}{f_x(x,y)} \tag{4}$$

The gradient histograms are then generated for each cell within the image. The interval of $[0,\pi)$ is evenly divided into nine bins. This value chosen as the number of orientations is offered by Dalal and Triggs [5] to result in better human detection. The association of a bin with each pixel is based on the value of $\theta(x,y)$ and is weighted by the value of $m(x,y)$. Two nearest bins to the value of $\theta(x,y)$ are updated to avoid aliasing effect.

The normalization process is the final step in HOG feature extraction. The blocks are the overlapping number of adjacent cells and usually consists of four cells, i.e., a set of 2x2 neighboring cells [5]. Feature vectors of the four cells within a block are accumulated to generate a normalization factor. This factor depends on the normalization scheme adopted for this step. We use *L1-sqrt* normalization scheme where ϵ is a small constant and *L1-sqrt* is defined as

$$L1\text{-}sqrt: \qquad \sqrt{\frac{\nu}{(\|\nu\|_1 + \epsilon)}} \tag{5}$$

With ν as the unnormalized feature vector, $\|\nu\|_k$, known as the *k-norm* of the vector is defined as:

$$\|\nu\|_k = \left(\sum_{i=1}^{n} |x_i|^k\right)^{\frac{1}{k}} \tag{6}$$

Once the features are extracted and normalized, a window of generated features is passed to the classifier to evaluate the presence of a specific object, i.e., human in the case of pedestrian detection. The concept of overlapping blocks and sliding detection window is depicted in Figure 2.

The classification stage is based on using a SVM classifier. The linear SVM is a discriminative binary classifier which is defined by the hyper-plane separating positive and negative regions. Given the training data together with their class labels, SVM constructs an optimum hyper-plane by its support vectors which define either a specific set of features belongs to a class of objects or not [6]. In its simplest form for two-dimensional feature space, SVM generates a line to divide positive and negative samples. SVM classifier looks for the answer of Equation 7 in a way that w is minimized so that *E(w)*, the total hinge loss, is minimized.

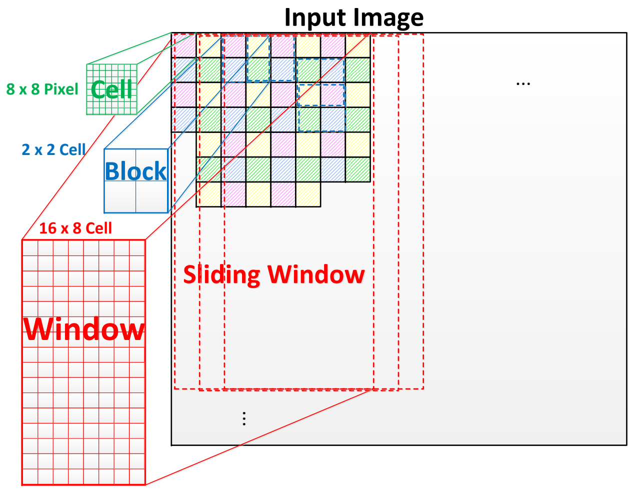

Figure 2. Cell, block, and sliding window in detection algorithm based on HOG and SVM. Detection window of 64x128 pixels, equivalent to 8x16 cells is used for pedestrian detection.

$$E(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} max\{0, 1 - y_i\langle w, x\rangle\} \tag{7}$$

During the detection and at the classification stage, linear SVM classifier compares the test data with the model data by calculating the dot product of the features vector x, and the weight vector w. The weight vector w is calculated and obtained during the training stage. A bias value b is also calculated during the training stage and is used in Equation 8 during the classification.

$$y(x) = w.x + b \tag{8}$$

The resulted *y(x)* defines whether the feature vector belongs to the specific class of objects or not by checking its sign as

$$\begin{cases} y(x) > 0 \Longrightarrow positive & (9) \\ y(x) < 0 \Longrightarrow negative & (10) \end{cases}$$

3.2. Hardware Implementation

3.2.1. HOG-SVM Hardware Accelerator

The pedestrian detection hardware accelerator includes two main stages of HOG extractor and SVM classifier as shown in Figure 3. The HOG feature extraction consists of a few computation stages. Figure 3 shows two different processing stages within the HOG extraction stage. The gradient calculation and histogram generation steps process the pixels within a cell, while the final step of block normalization modifies the generated histograms within a block. Processing the pixels within the cells has a different data access pattern than the normalization stage. This means some intermediate storage elements are required between these two functional blocks. Moreover, the histograms of cells are updated several times during the processing of different pixels within a cell or its neighbor cells. In our hardware implementation, the gradient calculation and histogram generation are merged into one module named as *HOGDescriptor*. Block normalization is kept as a separate module named as *HOGNormalizer*. Three different memory structures are considered at start, middle, and end of the implemented pipeline to address different data access requirements of each processing stage. The input image is read from off-chip memory and is passed through a line buffer, *ImageBuffers*, where four rows of the image are stored. This is because three rows of pixels are required for the calculation of $f_y(x, y)$. The other row is updated with new data which is required for the calculations in the next row of pixels.

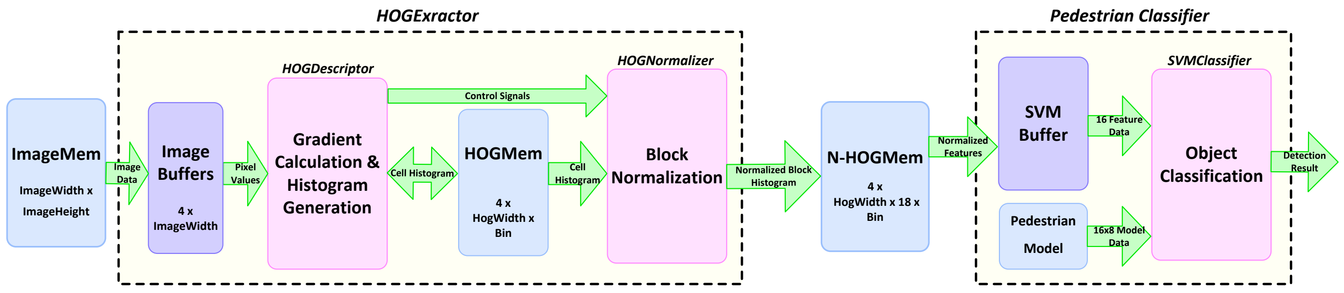

Figure 3. Block diagram of implemented hardware accelerator for pedestrian detection.

As discussed above, memory access could impose latency in the processing pipeline, especially in the case of off-chip memories. Furthermore, with the potential resource constraints for on-chip memories and the power/energy overhead, considerations should be taken into account for the wise and efficient use of storage elements. This could be addressed partly by defining the optimum memory access pattern within the algorithm.

The generation of histograms starts by gradient magnitude and direction calculation. The image pixels are read and gradient values are calculated starting from the top left of the image. Calculating the values for each pixel continues by scanning the image through each row. When it reaches the end of the row, scanning the next row is started. Keeping the pixel data from two adjacent rows as well as the data of the current row in the *ImageBuffers* results in accessing all the required data at the same cycle and improves the performance. This requires utilizing three line buffers and a shift register. An extra line buffer is considered to get updated for the calculation of the next row.

Implementation of *HOGDescriptor* is based on pipelined architecture and similar to the other related works, the floating point calculation and trigonometric computations are replaced by fixed-point calculation and lookup table (LUT) implementation. Calculation of gradients in *HOGDescriptor* is done in parallel for x and y directions. Once $f_x(x, y)$ and $f_y(x, y)$ are calculated, the magnitude and direction, as well as all distance weights required for bilinear interpolation, are calculated in parallel within the pipelined architecture to maximize the throughput. In this case, some Xilinx pre-generated IP cores are employed to provide a fully pipelined architecture for the mathematical functions used in the algorithm. The square root calculation is done by instantiating CORDIC IP core, which results in less resource utilization. The pipelined architecture of the *HOGDescriptor* results in generating one output value for the histogram in every cycle after passing the initial delay of about 100 clock cycles. It updates the value of histogram in *HOGMem* by calculating the gradient at every pixel within the image. Since the histogram of each cell is affected by the gradients in its neighbor cells, four histogram values within the *HOGMem* requires to get updated in each clock cycle. This could potentially put constraints on the throughput if not handled properly within the memory hierarchy.

The *HOGMem* is updated by the *HOGDescriptor* and accessed by the *HOGNormalizer* to provide a normalized version of the histogram across the blocks. For an optimal result, memory access requirement at both ends should be taken into account so that the feasibility of parallel and high throughput pipeline processing is obtained. Special memory pattern is defined for *HOGMem* where different cells within the image are divided into four different groups. Each cell is surrounded only by the cells belonging to the other groups. The pattern is shown in Figure 4 where *G1* to *G4* indices show the group for which the cell is assigned. Once the gradient at each point is calculated, its value will affect the histogram in four adjacent cells, and consequently, four histograms should be updated in *HOGMem*. The effect of one gradient on all adjacent cells is due to the spatial bilinear interpolation.

Update process requires four clock cycles unless the target memory places are accessible independently and in parallel, which is feasible by defining cell group division, as shown in Figure 4.

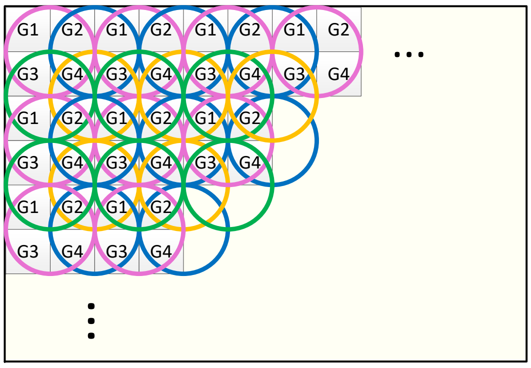

Figure 4. Division of cells into four different groups of *G1-G4* results in simultaneous access to all cells contributing in each HOG block.

On the other hand, *HOGNormalizer* requires accessing the value of four adjacent cells to generate a block and calculate the normalization factor for the block. In the defined pattern, it is guaranteed that each block, as defined in HOG algorithm, consists of four cells where each of them belongs to one group. In other words, such pattern assures that in any HOG block, only one cell from any specific group exists. As a result, *HOGNormalizer* is capable of accessing data from four separate memories in parallel to provide the values of the normalized histogram for all contributing cells of a block. By dividing the cells in this way and storing the value of cell histograms within a group in a separate memory, both functional blocks of *HOGDescriptor* and *HOGNormalizer* can update and access four different parts of *HOGMem* in parallel which reduces the total required processing time.

A similar pattern is defined for the memory storing the results of the *HOGNormalizer*, called *N-HOGMem*. As shown in Figure 4, each cell contributes to four different blocks by changing its local position within the block. Consequently, by the end of the normalization process, four different versions of a cell histogram are generated based on the normalization factors obtained through the formation of four different blocks containing each specific cell. In this case, four memory banks are associated with each group of *G1* to *G4*, to store four different versions of the cell histogram. Figure 5 shows the structure of *HOGMem* and *N-HOGMem* and their relation with the processing blocks within the *HOGDescriptor*.

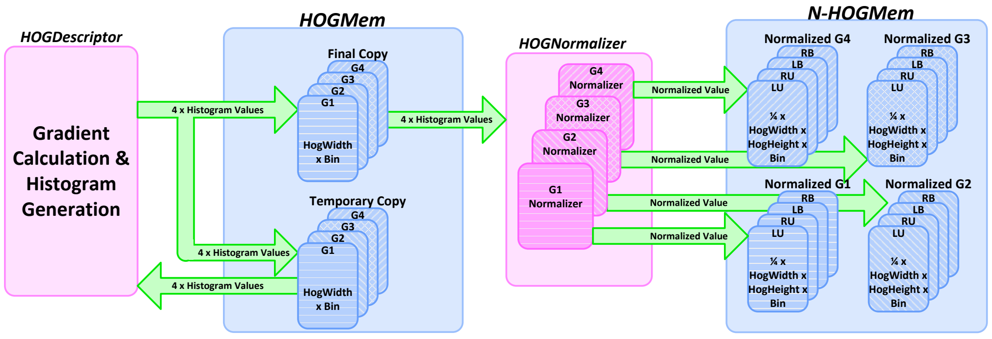

Figure 5. Memory hierarchy of *HOGMem* and *N-HOGMem*. The *HOGMem* consists of two copies update by *HOGDescriptor*. The final copy is accessed by *HOGNormalizer*. The *N-HOGMem* includes four instances of memory for each group cell, which is defined based on its position within the normalized block. The *LU*, *RU*, *LB*, and *RB* stand for left up, right up, left bottom, and right bottom positions correspondingly.

The final values of HOG descriptor are stored in *N-HOGMem* and *HOGMem* is just playing the role of an intermediate memory to store the un-normalized histograms. Hence, it is more efficient to reduce the memory size as much as possible to both satisfy its functional requirement and reduce the required resource usage. Considering the required time for processing of each block in *HOGNormalizer*, it is concluded that normalization of one row of HOG blocks is fast enough to be completed during the time one new row of cell histograms are updated completely in *HOGMem*. As a result, storing only four rows of cell histograms within the *HOGMem* and overwriting the same area once new cell histograms are calculated is sufficient.

Parallel and pipelined architecture is considered in the implementation of *HOGNormalizer* as well so that it does not impose any delay to the whole process. The calculated normalization factor of a block is used for normalization of all cells within the block at the same time in parallel. This approach results in data reuse and saves both power and time.

As shown in Figure 3, the last stage in pedestrian detection is the classification of HOG features through linear *SVMclassifier*. The access pattern at this stage differs from the order it is generated at the *HOGNormalizer* side. Therefore, the use of intermediate memory, *N-HOGMem* is inevitable. We show how the choice of compatible pipelined architecture for *SVMclassifier* helps in minimizing the size of *N-HOGMem* memory.

The classifier requires both the normalized feature data from *N-HOGMem* memory and the model data to calculate the dot product of them. The pedestrian model, resulted from off-line training process, is stored in a separate memory and accessed by *SVMclassifier* as shown in Figure 3. Each detection window for pedestrian detection is 16x8 blocks, and each block has a 36-element feature vector.

N-HOGMem provides access to 16 different HOG features through 16 memory banks. Even though 16 simultaneous data could be accessed in a cycle; these features do not provide one full column of the detection window. However, in two cycles, one feature for two columns of the window could be obtained from *N-HOGMem*. SVM classifier obtains the feature vectors of two columns every 72 clock cycles by circling through four different categories of feature data groups, i.e., *LU*, *RU*, *LB*, and *RB* shown in Figure 5. This is equivalent to accessing the feature vector of one column every 36 clock cycles when the buffers are full.

A parallel architecture matching with memory access is defined for the processing units in *SVMclassifier*. To increase the throughput as much as possible and satisfy the real-time requirements, we have defined a deep pipelined parallel architecture for the calculation stage. Maintaining the same processing speed at both stages of feature extraction and classification results in eliminating unnecessary storage elements. The data features for one column of the window are fed to the classifier. At the same time, the dot products are calculated by 16 different MAC units that are responsible for multiplication and accumulation required in the dot product. We name this processing unit *MACBAR*. Figure 6 shows MACBAR parallel architecture, which consists of 16 MAC units working in parallel, each fed with a model data and data feature separately.

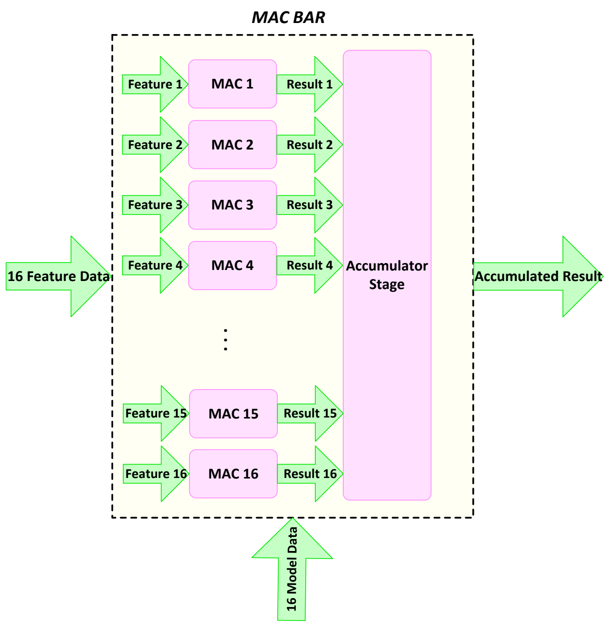

Figure 6. The *MACBAR* parallel architecture consisting of 16 *MAC* units.

Figure 7 shows the parallel and pipelined architecture of the SVM classifier with eight parallel *MACBAR* computation units. The feature data is fed to the classifier and pipelined through eight stages to calculate eight columns of eight different windows. Consequently, once all eight *MACBAR* units are filled with the data, the classifier calculates the SVM result of a window through 36 cycles. The detection window slides horizontally through the image until it reaches to the end of the row, when a new window starts from the next row. This approach results in classification speed exceeding the rate of feature extraction. This guarantees that the limited size of *N-HOGMem* considered in the design fully addresses our requirements. 285–293

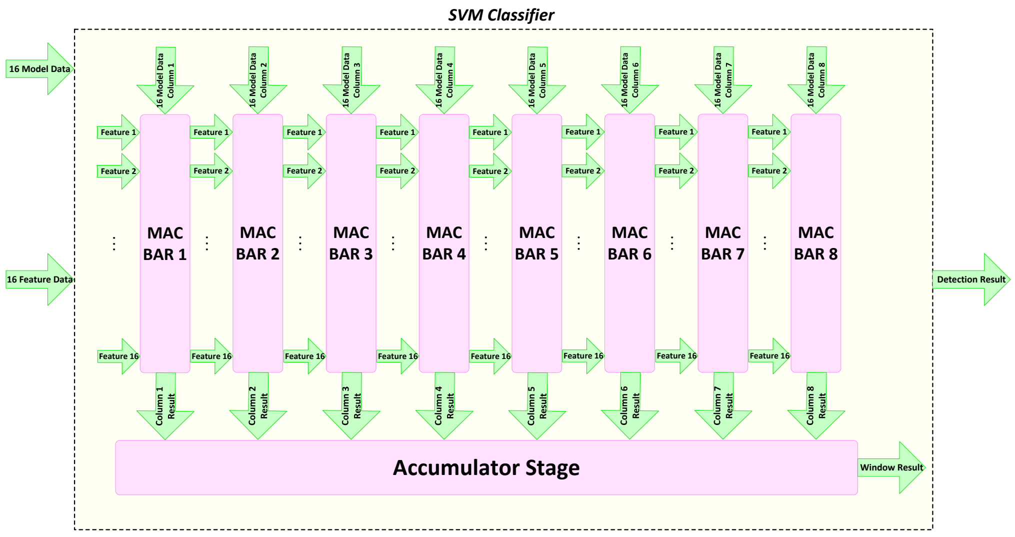

Figure 7. The parallel and pipeline architecture of *SVMclassifier* with 8 parallel *MACBAR* computation unit.

3.2.2. Multi-Scale Detection 294

Accurate and fast detection of pedestrians is one of the most challenging tasks of ADS. Humans with various sizes appear on the road at different distances from the car, 295–296 which results in the detection requirement of considering different sizes. Slight changes in human size are considered within the training stage by feeding variations of positive training samples to the SVM classifier. However, the classifier searches for a specific size of a human within its defined window size. Consequently, the presence of objects with a bigger or smaller size, which do not fit in the detection window is not achieved through the detection method shown in Figure 1. By use of down-sampled images at different scales detection of the objects with bigger size or farther distance to the car will become possible.

The image pyramid is generated by down-sampling the original image by various factors consecutively. The main parts of detection pipeline, including both HOG feature extraction and SVM classification, is then applied to each of the scaled images separately. The final result is achieved by merging all the detection results and choosing the detection with the highest probability based on the confidence score as generated by the classifier. The last part is usually handled by the non-maximum suppression (NMS) algorithm [15]. NMS considers the results of all windows which have an overlap of more than a specific value and then choose the window with the highest classification result to represent the final detection.

The real-time requirements of pedestrian detection in safety-critical applications such as ADS requires the employment of hardware accelerators within the detection pipeline. In the multi-scale detection scenario, where several scales of the image should be processed to check the presence of an object, utilization of hardware accelerator could be done in two different ways or a combination of both. Figure 8 shows two different ways of instantiating the hardware accelerator, which processes the HOG feature extraction and SVM classification.

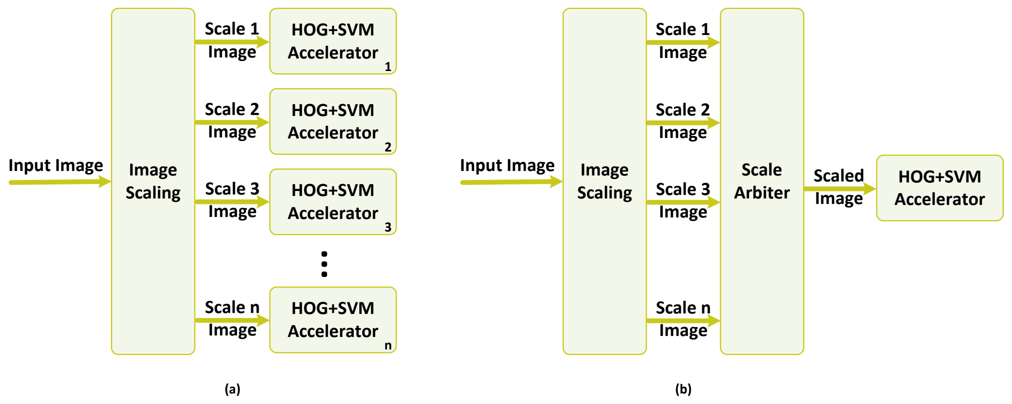

Figure 8. Potential hardware implementations based on conventional multi-scale detection approach. (a) represents the fully parallel approach and (b) represents the sequential approach.

In the first approach, shown in part (a), a functional block of image scaling reads an input image and generates various scales by down-sampling the original image. These scaled images are then processed in parallel by having several instances of the hardware accelerator working in parallel. This approach helps in maintaining the high throughput and real-time performance of the detection at the cost of higher resource utilization and consequently, higher power/energy consumption.

The second approach, depicted in part (b) of Figure 8, maintain the resource utilization as low as possible. The task of processing scaled images is handled by the same hardware accelerator, which is in charge of processing the original image. The *ScaleArbiter* functional block circulates through different images in the image pyramid and employ the same hardware accelerator by time multiplexing the access of different scales of the image to it. This approach benefits from low resource utilization; however, it lacks the capability of providing real-time performance and throughput as the processing time is increased by the factor of n.