BINARIZATION BASED IMPLEMENTATION FOR REAL-TIME HUMAN DETECTION

Shuai Xie, Yibin Li, Zhiping Jia, Lei Ju*

Shandong University of Computer Science and Technology
Jinan, China, 250101
e-mail: xieshuai1210@mail.sdu.edu.cn, {liyibing,jzp,julei}@sdu.edu.cn

ABSTRACT

Hardware implementation of human detection is a challenging task for embedded designs. This paper presents a real-time image-based field-programmable gate array (FPGA) implementation of human detection. Our implementation is based on the histograms of oriented gradients (HOG) feature and linear support vector machine (SVM) classifier. The novelty of this work is that we replace normalization process of HOG with a modified binarization process. Therefore, during classification process with SVM classifier, all multiplication operations are replaced by addition operations. All these modifications result in reduction of hardware resource. Experimental evaluation reveals that 293 fps can be achieved on a low-end Xilinx Spartan-3e FPGA. Moreover, a detection accuracy of 1.97% miss rate and 1% false positive rate is achieved. For further demonstration, a prototype system is developed with an OV7670 camera device. Restricted to the speed of camera, a detection rate of 30 fps is achieved.

1. INTRODUCTION

Real-time human detection is important for numerous application domains such as security, entertainment, surveillance and robotics. The histogram of oriented gradients (HOG) feature [1] is a widely used algorithm to address this problem. The histogram of oriented gradients is used for feature generation; linear support vector machine (SVM) classifier is used for classification. However, its high computational complexity makes it impossible to run HOG in an embedded CPU in real time.

In the past few years, several FPGA implementations have been presented. Kerhet et al [2] presented a human detection system based on a SVM-like algorithm. A detection rate of 15 fps for a 15×30 image was achieved with a power consumption of 500mW. In paper [3], a hardware implementation of the HOG generation process (without classification process) was presented. This work achieved 30 fps for an image of 640×480. An implementation of human detection [4] using HOG and AdaBoost classifier was presented. A binarization process [5] and pipeline architecture were introduced to reduce the memory utilization. Theoretically, 112 fps was achieved with a Virtex-5 FPGA. With simplifications, roughly 16% drop in detection accuracy was suffered. In paper [6], an implementation based on the original HOG and SVM algorithm was presented without any simplification. With single precision floating point, this implementation was developed on a Virtex-6 FPGA. This implementation is able to process VGA stream with 60 fps. For those implementations of HOG based human detection, much hardware resources are consumed that only high-end FPGA devices can afford. Also, the detection accuracy is reduced severely in some cases. Therefore, previous works evidently fail to meet the requirements of embedded applications with constrained resources.

In this paper, we present a real-time human-detection implementation targeting low-end FPGA devices. In this implementation, normalization process of HOG is replaced by a modified binarization process. Therefore, during classification process with SVM classifier, all multiplication operations are replaced by addition operations. Besides, a pipeline architecture is introduced to accelerate processing rate. With all these modifications, our implementation can be developed on a low-end Xilinx Spartan-3e FPGA device with a processing rate of 293 fps. Moreover, a detection accuracy of 1.97% miss rate and 1% false positive rate is achieved.

2. THE HUMAN DETECTION ALGORITHM

The human detection algorithm consists of two parts: the HOG feature generation process and linear SVM classifier classification process. HOG feature is used to characterize a given image, and linear SVM classifier is used to make a classification.

The original HOG and SVM algorithm can be divided into four steps:

- Gradient and direction calculation;
- Histograms generation;
- Normalization.
- Classification

Computation-intensive operations are mainly executed in the first and third steps. To implement this algorithm on a low-end FPGA device, a modified binarization process is adopted in place of the normalization process. As a result, during the classification process, all multiplication operations are replaced by addition operations.

In the followings, the details of the modified algorithm are explained.

978-1-4799-0004-6/13/$31.00 ©2013 IEEE

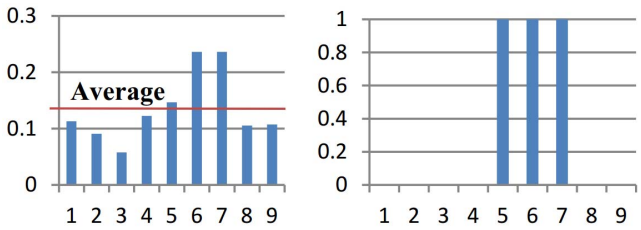

Fig. 1. An example of binarization

A. Gradient and Direction Calculation & Histogram Generation

The HOG parameters used in this study are the same as those in [1]. Gradient and direction are calculated by the following equations:

$$\begin{cases} f_x(x,y) = f(x+1,y) - f(x-1,y) \\ f_y(x,y) = f(x,y+1) - f(x,y-1) \end{cases} \quad (1)$$

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

$$\theta(x,y) = \tan^{-1}\frac{f_y(x,y)}{f_x(x,y)} \quad (3)$$

where *f(x,y)* denotes the luminance at *(x,y)*, *m(x,y)* is the gradient value, *θ(x,y)* is the direction at *(x,y)*. For color image we used, gradient *m* of each color channel is separately calculated, and only the gradient *m* with the maximum luminance is stored.

To compute gradients of orientation histogram, orientation is divided into 9 bins (20 degree each). For each pixel of a cell, two weighted votes are calculated for the nearest bin and the bin which the pixel belongs to. The vote is based on gradient magnitude *m*, whereas the weight is calculated according to direction *θ*. This process is calculated by the equations:

$$m_n = (1-\alpha)\,m(x,y) \quad (4)$$

$$m_{nearest} = \alpha m(x,y) \quad (5)$$

$$\alpha = \frac{9\theta(x,y)}{\pi} - (n+0.5) \quad (6)$$

In our implementation, standard RGB-565 image is used. Only high 4 bits of each channel are used for calculation, which means there are only 512 different values of gradient *m* and direction *θ*. The weight vote of each pixel can be calculated in advance and pre-stored in a look-up-table with 1 KB BRAM.

Then, the votes of a cell are summed up according to their directions. Finally, the histogram is generated for each cell.

In our implementation, each block contains four histograms, which is a nine-dimension vector.

Table 1. Parameters

Input image	320×240 pixels
Detection windows	64×128 pixels
Cell	8×8 pixels
Block	2×2 cells
Step stride	8×8 pixels
The number of bins	9

B. Normalization

The next process is normalization. In Dalal's paper, it was proved that the L2-norm has the best performance. This process is performed using Formula 7:

$$v = \frac{V_k}{\sqrt{V_k^2 + \varepsilon}} \quad (7)$$

V_k denotes the feature vector of the *k*-th block, ε is a constant used to prevent the denominator from being 0, *v* is the normalized feature vector.

This process contains square, square root, and division operations, thus making compact FPGA implementation difficult.

Our implementation adopted a binarization process. Comparing with Negi's implementation, for each block the average value of HOG features is calculated and then set to the threshold. As shown in Fig. 1, final HOG feature is set to 1 if it is greater than the average value; otherwise, it is set to 0. After this process, the memory cost of each HOG feature is reduced. Moreover, regardless of whether there is a normalization process, the generated HOG features stay same. With this modification, normalization process becomes unnecessary.

By adopting this binarization process, both memory utilization and computational complexity reduce.

C. Classification Process

We use Dalal's INRIA pedestrian database [1] to train a linear SVM classifier offline with our modified HOG feature algorithm. Given the used parameters, the generated classifier is a 3781 dimensions vector. The classification process is performed by multiplying first 3780 elements of the classifier and corresponding HOG features, then adding up the results of the multiplication and the 3781-th element of the classifier. Comparing the result with threshold, the classifier can determine whether there is a target in the detection window.

With the binarization process, each HOG feature has a value of 1 or 0. The classification process can be performed by adding up the elements of the classifier with a corresponding HOG feature of 1. Statistically, 40.5% of the HOG features are 1, which indicates that, for each classification process, roughly 1531 addition operations are

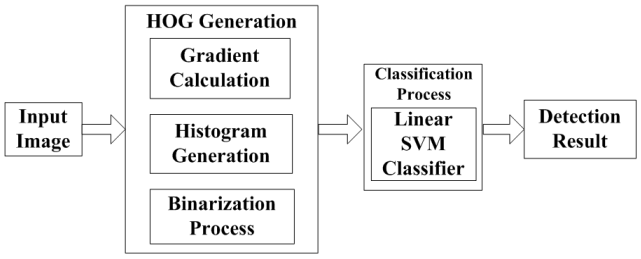

Fig. 2. The structure of human detection system

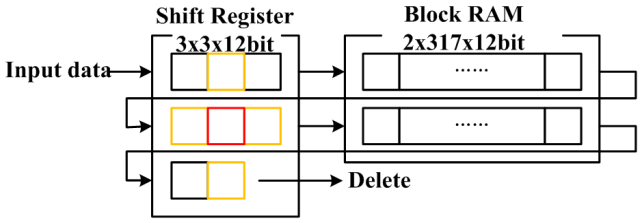

Fig. 3. Stream structure for the calculation of gradient and direction

needed instead of 3780 multiplication operations.

3. FPGA IMPLEMENTATION

We use an OV7670CMOS video camera as input device, which provide 30 frames of 320×240 pixels per second. The used HOG parameters are shown in Table 1. As shown in Fig.2, the hardware architecture is divided into two parts: the HOG generation process and the SVM classification process.

In the following, the details of hardware architecture and data stream for each process of the HOG and SVM algorithm are explained.

A. The HOG Generation Process

To accelerate the classification process, a pipeline architecture is adopted. As shown in Fig. 3, for gradient and direction calculation, three lines of three-stage shift registers are used to store four adjacent data. Two-line BRAM is used to store other 317 values. As mentioned before, the calculation of m_n and $m_{nearest}$ is performed by a look-up-table. Similar pipeline architecture is used for the histogram generation, binarization, and SVM detection processes.

After previous process, we get the weighted votes of each pixel. A histogram of cell is generated by summing up the votes of one cell. Similar to Fig.3, partial histogram is calculated for every 8 pixels of one line and stored in a temporary register. Then the stream of partial histograms is loaded into BRAM, so that partial histograms of 8 lines are eventually summed up. Finally, histogram for each cell is generated.

In our implementation, we adopted the optimized binarization process in place of normalization process. This process needs the adjacent feature vectors of four cells;

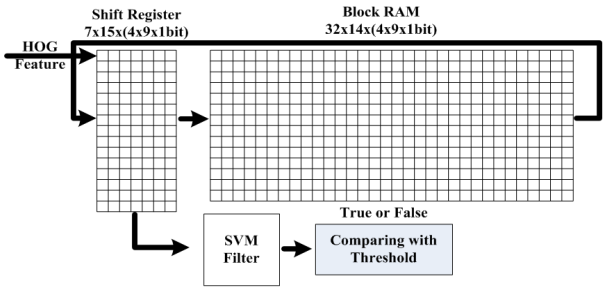

Fig. 4. Hardware structure for classification process

therefore, two lines of two-stage shift registers are used to store the feature vectors of four cells which are need to be calculated now and one-line of BRAM buffers are used to store the feature vectors . The average value of each cell's feature vector is calculated and cached in a temporary register along with the feature vector of this cell. Every time new feature vector is streamed in, the average value of each block is calculated. Then each feature value is coded in binary mode.

B. Classification Process

With binarization process, the classification process can be performed by adding up the elements of the classifier with a corresponding HOG feature of 1. As shown in Fig. 4, we build a 3780 dimensions filter to store the elements of the classifier. When the HOG features of one detection window are loaded into the filter, the elements, which have a corresponding HOG feature of 1, are added up. As shown in Fig. 4, for the classification process, fifteen lines of seven-stage shift registers are used to store HOG features of detection window. Fourteen lines BRAM buffers are built for other features.

4. IMPLEMENTATION RESULT AND EVALUATION

The proposed human detection was developed in Verilog-HDL and synthesized for a low-end Xilinx Spartan-3e XC3S500E device [7] with Xilinx ISE 13.1 tools. To train the classifier, HOG features were generated by a modified OPENCV [8]. Using Libsvm [9] library of MATLAB, linear SVM classifier is trained offline, and detection accuracy is evaluated.

In our implementation, we adopt a modified binarization process in place of normalization process. With this process, classification process can be implemented by addition operation. Moreover, a pipeline architecture is introduced to accelerate the processing rate. All those modifications result in acceleration of detection speed with less hardware resource and power consumption.

Comparing with our implementation, Komorkiewicz [6] has implemented the original HOG algorithm by using single precision 32-bit floating point values for high

Table 2. The results of FPGA implementation

	Our work on Spartan-3e	Neqi's work [4] on Virtex-5	Komorkiewicz's work [6] on Virtex-6
SLICE	2041	2,181	32,428
LUT	3,379	17,383	113,359
FF	2,602	2,070	75,071
BRAM	6	36	119

detection accuracy. But it used much more hardware resources and only can be mapped on a high-end FPGA such as Virtex-6. In Negi's work [4], a binarization process was used with a constant threshold. Therefore, the normalization process was necessary. Moreover, every 8 1-bit HOG features were considered as one 8-bit feature during the classifier training and classifying processes. Those modifications resulted in a reduction of resource utilization, but a 16% drop in detection accuracy was suffered.

The implementation results of ours, Negi's [4] and Komorkiewicz's [6] works are listed in Table 2. Clearly, the resource utilization of our implementation is lower than other two implementations.

With a maximum frequency of 67.75MHz, a detection speed of 293 fps can be achieved. This detection speed is approximately 2.6× faster than Negi's 112 fps and 139.5× faster than software implementation on a PC with an Intel Core E6550 processor. A prototype system with an OV7670 camera is developed for further demonstration. Although restricted to the speed of this camera, a detection speed of 30 fps was achieved. This speed can satisfy the real-time demands of embedded applications.

For comparison, we re-implement Negi's work on software. The INRIA pedestrian database is used to train classifier and test detecting accuracy. The test results are summarized as a detection error trade-off (DET) curve in Fig. 5. Compared with Negi's implementation, our work achieves a detection accuracy of 1.97% miss rate and 1% false positive rate. Although it is worse than the original algorithm, it's better than Negi's implementation and can satisfy the demands of embedded applications.

5. CONCLUSION

In this paper, we presented a real-time low-power implementation of human detection with HOG feature and linear SVM. Using a low-end Xilinx Spartan-3e FPGA, our implementation could achieve a detection speed of 293 fps. While some simplifications were made to alleviate hardware complexity, a detection accuracy of 1.97% miss rate and 1% false positive rate was achieved. This detection accuracy could satisfy most demands of embedded applications.

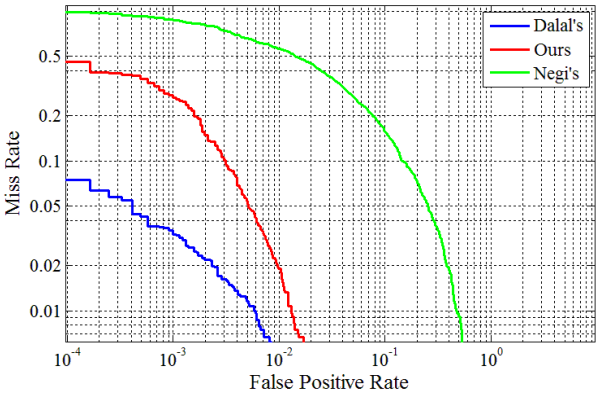

Fig. 5. DET curve for detection accuracy

6. REFERENCES

[1] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Proceedings of the 2005 International Conference on Computer Vision and Pattern Recognition*, vol. 2, 2005, pp. 886-893.

[2] A. Kerhet, F. Leonardi, A. Boni, P. Lombardo, M. Magno, L. Benini, "Distributed video surveillance using hardware -friendly sparse large margin classifiers," *AVSS* ,2007, pp.87-92.

[3] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, Y. Nakamura, "Hardware Architecture for HOG Feature Extraction," *Intelligent Information Hiding and Multimedia Signal Processing*,2009, pp.1330-1333.

[4] K. Negi, K. Dohi, Y. Shibata, K. Oguri, "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm," *Field-Programmable Technology (FPT), 2011 International Conference on. IEEE*, 2011, pp.1-8.

[5] W. Sun and K. Kise, "Speeding up the Detection of Line Drawings Using a Hash Table," in *Proceedings of The 1st China Japan Korea Joint Workshop on Pattern Recognition*, vol. 2, nov 2009, pp. 896–900.

[6] M. Komorkiewicz, M. Kluczewski, M. Gorgon, "Floating point HOG implementation for real-time multiple object detection," *Field Programmable Logic and Applications (FPL)*, 2012, pp.711-714.

[7] Xilinx, "Spartan-3 Generation Configuration User Guide", 2009.

[8] G. Bradski and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library. Cambridge, MA: O'Reilly,2008.

[9] Chih-Chung Chang, Chih-Jen Lin, "LIBSVM: A library for support vector machines," *ACM TIST (TIST)*, vol. 2, issue. 3, no.27, 2011.

[10] Xilinx, "Virtex-5 FPGA User Guide", 2012.