Data Reduction and Parallelization for Human Detection System

Mao Hatto [†], Takaaki Miyajima [‡] and Hideharu Amano [†]

[†] *Graduate School of Science and Technology, Keio University, Japan*
asap@am.ics.keio.ac.jp

HOG (Histogram of Oriented Gradients) is one of the effective ways for extracting feature values. Also, Real Adaboost algorithm has high recognition ratio, and it is adequate to hardware implementation. Many researches on human detection systems adopted these two algorithms and had achieved progress. However, data volume of HOG feature is still a problem in the whole system. Data volume from only one frame could be over 1 GB, and this data volume causes some difficulties from the view point of both sending data to a server and execution speed. Especially, since much data volume presses also internal data communication between modules in hardware execution, much data volume could be a bottle-neck of the whole system operating speed.

Here, a high speed and small memory consuming implementation of human detection system using Hardware-Software Co-design is proposed. For the executing speed of the system, HOG feature values are accelerated by an FPGA, and Real Adaboost detection is executed only by accessing ROM data in the FPGA. As a result, HOG+Real Adaboost part was accelerated about 23.1 times faster compared to the software execution. Whole system had been implemented on a single board, and it achieved 3.22 times speed up from camera input to VGA display output. Also we tried to reduce feature data volume, and achieved 93.75% of data compression compared to double precision calculation, with only 2.68% loss of the recognition accuracy.

I. Introduction

As technologies of high performance computing and pattern recognition have developed rapidly, a human detection system has also gathered attention recently. Human detection systems can be applied for many objectives, such as an automobile safety system, an automatic traffic census system and a crime prevention system. In these systems, collected feature data are expected to be stored in data base servers, and be used for big data analysis.

Although various kind of methods for extracting feature data for human detection system are proposed, HOG (Histogram of Oriented Gradients)[1] proposed by N.Dalal and B.Triggs has been an effective way for extracting feature values. Also, Adaboost algorithm[2] proposed by R.E. Schapire and Y.Singer has high recognition ratio and is easy for hardware implementation. Furthermore, Real Adaboost[3], an improved algorithm extends output values of weak classifier from a binary number to a real number.

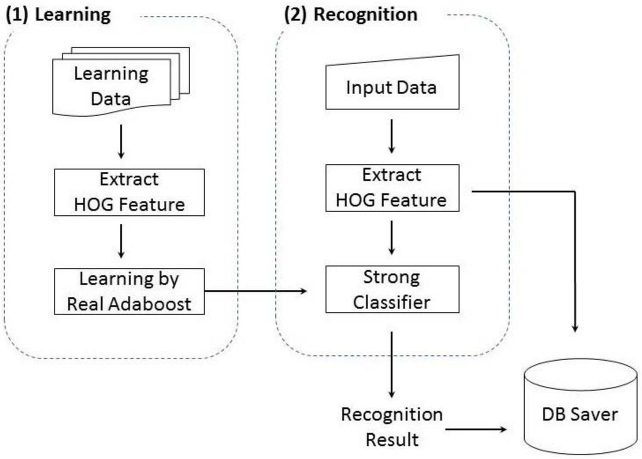

Fig. 1: Overview of Human Detection System

Figure 1 shows the overview of a human detection system. It can be divided into two phases: learning phase and recognition phase. The learning phase extracts HOG feature values from prepared learning positive and negative data, and performs learning using Real Adaboost Algorithm. This phase generates strong classifier, and it is used in the recognition phase. In the recognition phase, extracting HOG feature values from input data are judged whether the input data include humans or not. If the system detects human in the picture, extracted HOG values are sent to the database server.

A lot of researches on human detection systems adopted these two algorithms, and achieved a certain progress. However, data volume of HOG feature is still a problem of the whole system. Although one of the related work proposed feature data reduction using binary pattern for HOG feature values[4], it also causes deterioration of recognizing accuracy. As performance of detection is the most important evaluation criteria, it should be kept even data values are reduced.

This paper tries to accelerate a human detection system through feature data reduction using Hardware-Software Co-design. For acceleration, FPGA implementation with parallel execution is promising. In the implementation, both high performance and high data compression were tried to be achieved while keeping the resource utilization and recognition ratio.

This paper is organized as follow. Section II gives an overview of a human detection system using HOG feature and Real Adaboost, then Section III mentions the related works. Section IV explains implementation on FPGA, and Section V discusses data volumes and how to reduce them. Section VI shows the result of evaluation, and VII concludes the paper.

II. Human Detection System

This section gives an overview of the human detection system, and explains algorithms used in there.

A. Feature Extraction

Although various feature extraction methods were proposed, HOG (Histogram of Oriented Gradients) has been the most popularly used method in object detection systems. In HOG feature extraction, input window is divided into some cells, which are composed by a certain number of pixels. Gradients histograms are generated in each cell, and they are normalized in each block, which contains some cells. More details of HOG algorithm is shown below.

1) Gradient Computation: The first step of calculation is computing Intensity derivative f_x, f_y by intensity $L(x,y)$ in each pixel. Then, intensity gradients $m(x,y)$ and direction of gradient vector $\theta(x,y)$ are computed.

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

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

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

2) Orientation Binning: The next step is making histograms in each cell. Calculate *bin* for binning by intensity gradients $\theta(x,y)$ and azimuthal quantum number N.

$$bin = Round(\theta(x,y) + \frac{\pi}{2}) \times \frac{180}{\pi} \times \frac{N}{180} \tag{4}$$

3) Block Normalization: Finally, feature values v are normalized in each block.

$$v_n = \frac{v}{\sqrt{(\sum_{i=0}^{k} v(i))^2 + \epsilon}} \tag{5}$$

B. Recognition and Classification

Adaboost, a machine learning algorithm proposed by Y.Freund and R.Schapire, has high degree of detective ratio and is easy to be implemented. While Adaboost algorithm uses weak classifier with binary {+1, -1} outputs, Real Adaboost, an improved version of Adaboost, uses weak classifier with real number outputs. Compared with Adaboost, Real Adaboost has high recognition ratio with fewer classifiers.

Specification of Real Adaboost algorithm is shown below.

As a precondition, N learning sample data $(x_i, y_i), (i = 1, ..., N)$ and weighting number $D_t(i)$ are prepared. x_i means input sample data and y_i shows class label which shows whether any person appears in the sample data or not. D_i is initialized $1/N$ at the beginning of the learning phase.

The first step of learning generates probability density distributions. By feature values of learning sample x_i and quantifying number j, two probability density distributions W_+^j, W_-^j are generated.

$$\begin{cases} W_+^j = \sum_{i:\ j \in y_i = +1} D_t(i) \\ W_-^j = \sum_{i:\ j \in y_i = -1} D_t(i) \end{cases} \tag{6}$$

Next, compute candidates of weak classifier $h(x)$ and select proper weak classifier by evaluation value Z_m.

$$h(x) = \frac{1}{2} ln \frac{W_+^j + \epsilon}{W_-^j + \epsilon} \tag{7}$$

$$Z_m = 1 - 2\sum_j \sqrt{W_+^j W_-^j} \tag{8}$$

Finally, strong classifier is created by adding all classifiers. λ means a threshold value. If $H(x)$ is positive, it means that human exists in the window.

$$H(x) = sign\left\{\sum_t h_t(x) - \lambda\right\} \tag{9}$$

III. Related Works

There are three important measures in a human detection system: detection accuracy, detection speed and data volumes.

To improve the detection accuracy, combining some feature values is an efficient way. J.Yao, et al, proposed a human detection system using Covariance features[5], which focuses on spatial variation from background difference. P.Ott, et al, proposed Color-HOG[6], which focuses on color information.

In order to accelerate detection speed, many approaches using hardware architecture are proposed. M.Hahnle, et al. implemented pedestrian detection using HOG and Support Vector Machine (SVM) on an FPGA[7], and they achieved processing of 64 high resolution images (1920x1080 pixels) per second. R.Benenson, et al. accomplished 100 frames per second on a single CPU and GPU system[8].

Also for reducing the data volumes, C.Matsushima, et al. proposed binarized HOG feature on human detection system using HOG and Real Adaboost[4]. Binarizing HOG feature using equation (10) reduced feature data to 1/64. v_d and b_d represent HOG feature and binarized HOG feature respectively, and *th* means threshold value which is set in 0.03 in this proposal. Although this data reduction method declines detection accuracy about 10%, they proposed binary selection algorithm to keep detection accuracy as same level as conventional way.

$$b_d = \begin{cases} 1 & v_d \geq th \\ 0 & otherwise \end{cases} \tag{10}$$

IV. Implementation

A. Algorithm

Listing 1 shows C-like pseud code of our human detection system. *Gray_scaling()* converts input data to gray scaled picture, and *initialization()* allocates memory and initializes their value.

Though block window is shifted in a target window in the HOG calculation, processing unit accesses same place of pixel data. As this increases the number of memory access, it is better to calculate all gradients values before extracting the target window. *Gradient_computation()* and *Orientation_binning()* execute pre-calculations of Equation (2) and (4).

In *for* block, target windows are extracted by Raster Scan. *Load_Histogram()* fetches histogram values from DDR memory, and *Normalization()* executes Equation (5). Detecting results by *RealAdaboost()* are stored into an array by *Voting()*.

After the raster scanning, detected windows are integrated by *Mean_Shift()* and *Nearest_Neighbor()*, and *Output_image()* stores output image data to DDR, then finally these data are sent to VGA display.

Listing 1: Pseud Code of Human Detection System

```
int main() {
    Input_image();
    gray_scaling();
    initialization();
    // Calculate gradients before extract window.
    Gradient_computation();
    Orientation_binning();
    for(Raster Scan){
        // HOG Extraction
        Load_Histogram();
        Normalization();
        // Detecting by Real Adaboost
        if(Adaboost() == true)
            Voting();
    }
    Mean_Shift();
    Nearest_Neighbor();
    Output_Image();
}
```

B. Profiling the whole application

Table I shows profiling result by ARM Coretex-A9 processor on ZedBoard. As top four routines occupy about 96% of whole execution time, we focus on these routines. Since most of operations in *Load Histogram* and *Orientation Binning* are accessing to DDR3 memory, however, acceleration by FPGA cannot be expected much. Thus, we try to accelerate *Normalization*, *RealAdaboost* and *Gradient Computation* routines using an FPGA.

C. Software/Hardware Partitioning

Figure 2 shows software and hardware partitioning based on Listing 1.

First, camera module inputs image data into the FPGA through *Camera Interface*. ARM Coretex-A9 processor loads them and executes gray scaling and initialization. *HOG 1* module on programmable logic calculates gradient values, and sends back to ARM processor. *Normalization* and *Real Adaboost* logic also work as mentioned above, and finally *Display Controller* module loads output image data on DDR3 memory, then outputs to VGA display.

TABLE I: Result of Profiling

Sub Routine Name	# of Call	Execution Time	Ratio
Normalization	23,510	3.23 sec	42.27%
Load Histogram	23,510	2.04 sec	26.69%
Real Adaboost	23,510	1.86 sec	24.29%
Orientation Binning	1	0.24 sec	3.15%
Gradient Computation	1	0.18 sec	2.39%
Gray Scaling	1	0.047 sec	0.61%
Window Merging	1	0.015 sec	0.19%
Initialization	1	0.13 sec	0.17%
Voting	16	0.0091 sec	0.12%
Input image	1	0.0086 sec	0.11%
		7.64 sec	100%

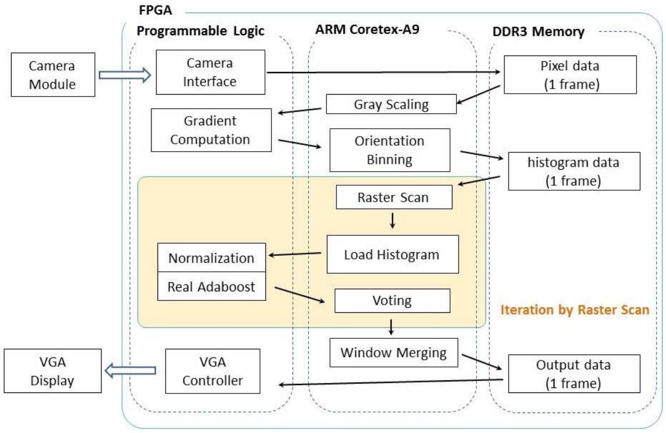

Fig. 2: Software and Hardware Partition

D. Architecture

Figure 3 shows the whole architecture in the FPGA. There are three accelerated modules, shown by light-yellow, are mapped on Programmable Logic, and they are connected to ARM Coretex-A9 processor. AXI DMA Engines are used for converting AXI4 (Memory Mapped communication) to AXI4-Stream communication.

More specifications of these logic are mentioned below.

1) Gradient Computation: For 18 bit x 1 frame (800x600 pixels) input, 14 bit x one frame is outputted. *Gradient Computation* module, shown in Figure 4, operates as *Gradient_computation()* routine in Listing 1. One input data composed of two Intensity derivative (f_x, f_y) are pushed into a FIFO. These input data are calculated as Equation (1) to (4), then the pair of intensity gradient $m(x, y)$ and direction of gradient *bin* are sent back to ARM core.

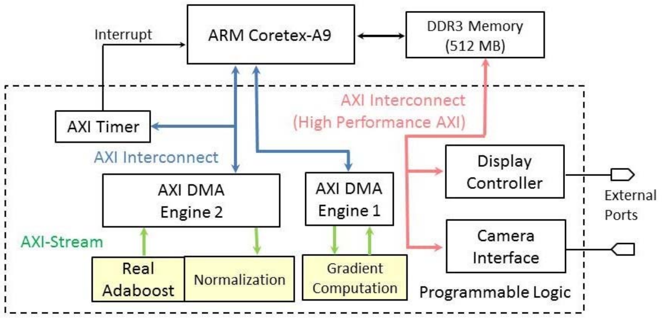

Fig. 3: Whole Architecture

Bit width of each signal is shown in red letter, and *SQRT* and *Arctan*, which are shown in light-yellow box, are generated by Xilinx Core generator.

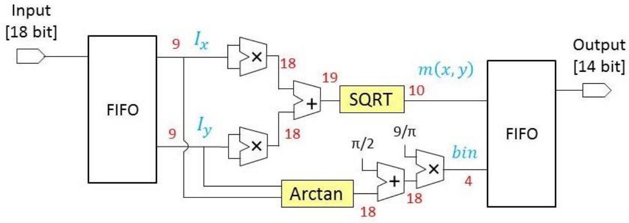

Fig. 4: Block Diagram of Gradient Computation

2) Normalization: For 13 bit x 1152 inputs, 6 bit x 3780 frame are outputted. This module normalizes each block in input window. One window has 8x16 cells, and each cell has 9 features. Thus, there are 1152 inputs. Also, as we set 2x2 cells in one block, there are 3780 output data.

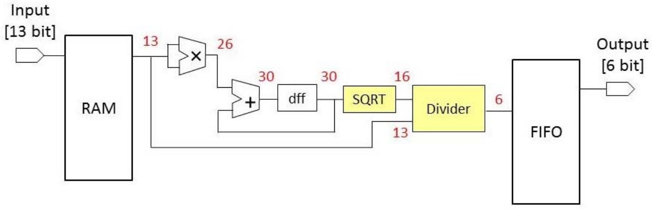

Fig. 5: Block Diagram of Normalization logic

3) Real Adaboost: Figure 6 shows block diagram of Real Adaboost module. As the only execution in this module is accessing ROM data which has previous classifier calculation data, block diagram of this module is simple. After accessing the ROM, all of these data are summed up and compared with λ, a threshold for detection.

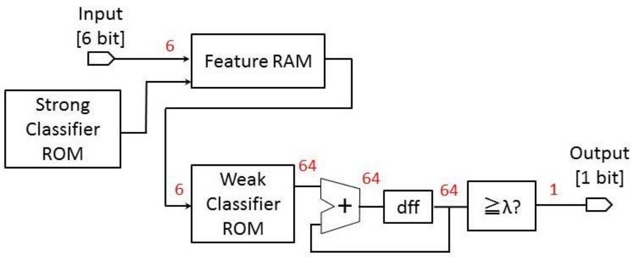

Fig. 6: Block Diagram of Real Adaboost module

E. Implementation Environment

Table II shows the environment of implementation. Target FPGA is ZedBoard by Avnet inc., and we used ARM Coretex-A9 as a processor with the FPGA. ARM Coretex-A9 has 512 MB DDR3 memory, and it operates up to 667 MHz.

Also, we adopt MT9D111 Camera Module by Micron Technology, Inc. This camera module generates 800x600 pixel data on 30 fps. Pixel data format from the camera module is expressed in 565 RGB, it uses 16 bits in each pixel data.

TABLE II: Implementation Environment

Target Device
ZedBoard(XC7Z020-CLG484-1, Avnet,Inc.) Software Processor ARM Coretex-A9(Up to 667 MHz operation) Memory: 512 MB DDR3 memory
Camera Module
MT9D111 Camera Module (Micron Technology, Inc.) Input image: 800x600 px Frame ratio: 30 fps Data format: 565 RGB

V. Data Reduction

HOG feature has N feature values in each cell (N represents an azimuthal quantum number in equation (4)), and also these feature values are normalized in shifted block. If we assume that one window has 8x16 cells, one block composed by 2x2 cells and azimuthal quantum number is 9, there are 3780 (7x15x2x2x9) feature data generated in total. In software execution, HOG feature values are calculated in double precision (8 byte), thus, total data volume per window becomes 30.24K Byte.

As detection window slides and also window size is scaled in the recognition phase, total occupied memory space is increased. If we assume that scanning 50,000 windows per one input image, total amount of memory space exceeds 1.5 GB. Using such a huge memory space in an FPGA, we need external memory like DDR3 SDRAM. It means communication with DDR3 and programmable logic might be a bottleneck of execution speed. Also from the view of saving feature data to a data server, too much data volume increases its running cost.

As we showed in Section III, data reduction method from 64 bit to 1 bit using binarized HOG[4] had been proposed. Although this proposal reduced feature data volume much, detection accuracy is also decreased. If they kept detection accuracy as same level as conventional way by introducing binary pattern and binary selection algorithm, occupied resource and latency in the FPGA are increased.

In this paper, we propose an approach that is using fixed point calculation to reduce feature data volume without large additional resource and complex coding.

A. Data Reduction Method

In the classification phase of Real Adaboost, the accuracy of weak classifier outputs has the highest effects on the output accuracy, and classifier outputs are calculated by HOG feature value (Equation (7)).

Considering the above, we calculated all possible outputs data of weak classifier (Equation (6)) in a high precision in advance, and storing them into the ROM on the FPGA. Therefore, in the FPGA, we need to access the ROM only. As the probability density is distributed in a weak classifier binning in 64 (2^6), bit width for accessing the ROM is 6 bits. This method can decrease HOG feature data from 64 bits to 6 bits, with the same accuracy as software execution in double precision. Figure 7 compares original Real Adaboost classification algorithm and ours.

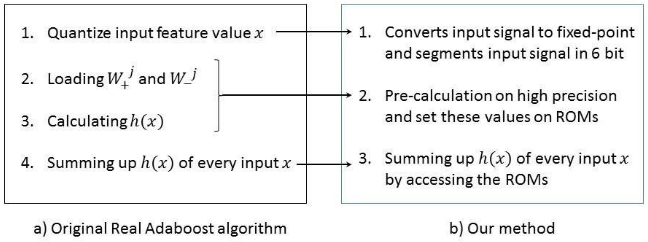
1. Quantize input feature value x
2. Loading W_+^j and W_-^j
3. Calculating $h(x)$
4. Summing up $h(x)$ of every input x
1. Converts input signal to fixed-point and segments input signal in 6 bit
2. Pre-calculation on high precision and set these values on ROMs
3. Summing up $h(x)$ of every input x by accessing the ROMs
a) Original Real Adaboost algorithm
b) Our method

Fig. 7: Data reduction method

VI. Evaluation

A. Resource Utilization

Table III shows resource utilization in the FPGA. As the slice register and LUTs are 19% and 27% respectively, there is a room to increase modules for parallel processing in the future extension.

TABLE III: Resource Utilization

Logic Utilization	Used	Available	Utilization
Number of Slice Register	20,648	106,400	19%
Number of Slice LUTs	14,821	53,200	27%
Number of Block RAM	99	420	24%
Number of bonded IOBs	30	200	15%
Number of RAMB36E1/FIFO36E1s	27	140	19%
Number of RAMB18E1/FIFO18E1s	72	280	25%
Number of DSP48E1s	13	220	5%

B. Execution speed

1) Hardware Operating Speed: From the implementation report, FPGA operates in **120.7 MHz**. Since Gradient Computation takes 960,600 clock cycles per frame and Normalization + Real Adaboost takes 576 clock cycles per window, it takes 120m seconds per frame, which means 8.32 fps, when 23,510 windows are extracted in one frame. As only software execution takes 2.77 sec per frame in this case, our hardware modules operate in 23.1 times faster than software operation.

2) Whole System Operating Speed: Whole system operating speed, including communication between HW/SW, is shown in Table V. *Only SW* row shows operating time by only ARM Coretex-A9 processor on ZedBoard. *Gradient Computation*, including communication with Software, operates in 4.025 millisecond on our system, while operates in 19.66 millisecond on Software execution. *Normalization and Real Adaboost* operates 22.27 microseconds per window, while operates in 117.26 microseconds on Software execution. In total, our whole design achieved 3.22 speedup as software execution with 1,540 detecting windows per frame, which is same number of detecting windows with [4].

TABLE V: Operating Speed

	Only SW	Our design	speed up
Gradient Computation	19.66 ms/f	4.025 ms/f	4.89
Normalization+RealAdaboost	117.26 us/w	22.27 us/w	5.27
Whole system	2.93 s/f	0.91 s/f	3.22

C. Detecting accuracy

For objective estimation, we refer the classifier of Real Adaboost which is purveyed by Chubu University[10]. This classifier uses 2,054 positive samples and 6,258 negative samples, and learning in 500 times.

For test sample, we use 152 positive samples and 250 negative samples from INRIA Person Dataset[11]. Each positive sample has one pedestrian and each negative sample has no pedestrian.

Figure 8 shows Detection Error Trade-off (DET) curves of our system. DET curves plot false positives per window (FPPW) in x-axis and miss rate in y-axis. A perfect system with zero misses and false positives per window would be positioned at the origin, thus, closer to the origin means better classifier.

In Figure 8, testing result of original system which is operated by Software is shown by red solid line. Other dashed lines show the result of our system with adjusting bit width of output signal of Gradient Computation module. *integer* means output of *Gradient Computation* is truncated to integer value. *16bit fractional*, *8bit fractional* and *4bit fractional* mean output signal is rounded to each bit after the decimal point. As we can see from Figure 8, *16bit, 8bit and 4bit fractional* approximate to original system, while *integer* shows somewhat far from original system.

TABLE IV: Summary of Comparison with related works

	Device	Algorithm	frequency[MHz]	fps	windows/frame	Registers	LUTs	DSPs
[7]	Xilinx Virtex-5	HOG + SVM	270	64	-	42,987	38,535	357
[8]	Nvidia GeForce GTX 470	HOG + FPDW	-	135	-	-	-	-
[4]	Xilinx Virtex-5	Binarized HOG + Adaboost	44.85	62.5	1,540	2,181	17,383	-
[9]	Cyclone IV	HOG+SVM	40	72	-	34,403	23,247	68
Ours	Avnet ZedBoard	HOG + Real Adaboost	120.7	8.3	23,510	20,648	14,821	13

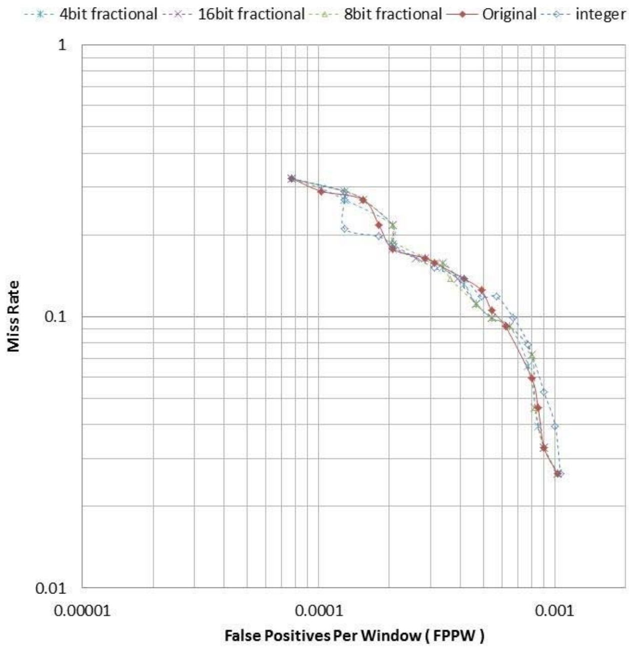

Fig. 8: DET curve of the system

D. Data volume

By using fixed point expressions, feature data are expressed by 6 bits. It means that the total compression ratio compared to the implementation using double precision floating point numbers is about 93.75%

In the proposed method, since the values of weak classifier outputs (Equation (7)) are calculated in advanced by double precision numbers and stored in the ROM on the FPGA, the accuracy is not much changed as that of the software execution as shown in the previous section.

Table VI shows that comparison between our method and related work. Binaryzed HOG[4] compressed HOG feature to 1/64 with accuracy deterioration of about 9.8%. In [4], they also proposed Integrating binaryzed HOG in order to prevent accuracy deterioration, however, it increases both the resource usage and the latency.

TABLE VI: Data volume comparison

	HOG	Our Method	Binalized HOG[4]
Data volume/window[Byte]	30,240	1968	472.5
Compression ratio	-	93.5%	98.4%
Accuracy deterioration	-	2.7%	9.8%

VII. Conclusion

We implemented a human detection system using HOG feature values and Real Adaboost detecting algorithm. In the human detection system, especially executing speed and feature data are important evaluation criteria. For the executing speed of the system, HOG feature values are accelerated by an FPGA, and Real Adaboost detection is executed only by accessing ROM data in the FPGA. Also we tried to reduce feature data volume, and achieved 93.75% of data compression compared with double precision calculation, without additional complex modification.

Table IV summarizes comparison with related works. Although other works accomplished faster fps than our system, our system extracts much window in one frame. Reducing feature data would be an important contribution because of these two viewpoints; storing to the data server and data parallelization in an FPGA. FPGA implementation using fixed point expression would be a good solution for making the system practical.

References

[1] N.Dalal and B.Triggs, "Histogram of Oriented Gradients for Human Detection," *IEEE Computer Vision and Pattern Recognition, vol.1, pp.886-893*, 2005.

[2] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, no. 55, pp. 119–139, 1997.

[3] R. E. Schapire and Y. Singer, "Improved Boosting Algorithm Using Confidence-rated Predictions," *Machine Learning*, no. 37, pp. 297–336, 1999.

[4] C. Matsushima, Y. Yamauchi, T. Yamashita, and H. Jujiyoshi, "A Method for Reducing number of HOG Features based on Real AdaBoost," *Information Processing Society of Japan SIG Technical Report, CVIM167*, 2009.

[5] J. Yao and J.-M. Odobez, "Fast Human Detection from Videos Using Covariance Features," *ECCV Visual Surveillance workshop (ECCV-VS)*, 2008.

[6] P. Ott and M. Everingham, "Implicit Color Segmentation Features for Pedestrian and Object Detection," *IEEE 12th Internalional Conference on Computer Vision*, 2009.

[7] M. Hahnle, F. Saxen, and M. Hisung, "FPGA-based Real-Time Pedestrian Detection on High-Resolution Images," *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2013.

[8] R. Benenson, M. Mathias, R. Timofte, and L. V. Gool, "Pedestrian detection at 100 frames per second," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2903–2910, 2012.

[9] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "Architectural Study of HOG Feature Extraction Processor for Real-Time Object Detection," *IEEE Workshop on Signal Processing System*, 2012.

[10] D. of Robotics Science and C. U. Technology, "Object Detection by Joint Feature Based on Relations of Local Features," http://www.vision.cs.chubu.ac.jp/jointhog/.

[11] N.Dalal and B.Triggs, "INRIA Person Dataset," http://pascal.inrialpes.fr/data/human/.