A Specialized Processor Suitable for AdaBoost-Based Detection with Haar-like Features

Masayuki Hiromoto  Kentaro Nakahara  Hiroki Sugano
Graduate School of Informatics, Kyoto University
Yoshida-hon-machi, Sakyo, Kyoto, 606-8501, Japan
{hiromoto,nakahara,hiroki}@easter.kuee.kyoto-u.ac.jp

Yukihiro Nakamura
Research Organization of Science & Engineering, Ritsumeikan University
1-1-1, Noji-Higashi, Kusatsu, Shiga, 525-8577, Japan
y-nakamr@fc.ritsumei.ac.jp

Ryusuke Miyamoto
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5, Takayamacho, Ikoma, Nara, 630-0192, Japan
miya@is.naist.jp

Abstract

Robust and rapid object detection is one of the great challenges in the field of computer vision. This paper proposes a hardware architecture suitable for object detection by Viola and Jones [9] based on an AdaBoost learning algorithm with Haar-like features as weak classifiers. Our architecture realizes rapid and robust detection with two major features: hybrid parallel execution and an image scaling method. The first exploits the cascade structure of classifiers, in which classifiers located near the beginning of the cascade are used more frequently than subsequent classifiers. We assign more resources to the former classifiers to execute in parallel than subsequent classifiers. This dramatically improves the total processing speed without a great increase in circuit area. The second feature is a method of scaling input images instead of scaling classifiers. This increases the efficiency of hardware implementation while retaining a high detection rate. In addition we implement the proposed architecture on a Virtex-5 FPGA to show that it achieves real-time object detection at 30 frames per second on VGA video.

1. Introduction

A recent challenge in the field of computer vision is object detection, which is often used for the detection part of face or pedestrian recognition systems. The object detection scheme proposed by Viola and Jones [9], one of the most widely used AdaBoost-based algorithms, achieves high detection rate and fast processing. They proposed a detection framework based on the AdaBoost learning algorithm using Haar-like features as simple edge detectors. This framework is used not only for face [2, 9] but also pedestrian [4, 11] recognition.

To realize practical systems for image recognition, real-time processing is indispensable, and several fast methods for object detection have been proposed to achieve such processing. For example, Intel® Open source Computer Vision library (OpenCV) [1], which provides many convenient libraries for popular image processing and computer vision algorithms on PC-based systems, can achieve rapid processing with highly optimized functions for Intel® CPUs using such techniques as SIMD operations or OpenMP interface. Since there is a limitation to accelerate processing with a single processor, further rapid processing is realized by implementation on parallel computing systems, such as multi-processors or multi-core processors.

On the other hand, real-time processing of image recognition is required for embedded systems, e.g., embedded automotive systems, security systems, or portable devices. For embedded use, we have to achieve real-time processing within the system’s limited power and size. Previous described methods, however, do not suit this purpose since they require high power consumption or large systems. Therefore specialized hardware, which consumes little power and can be built into small systems, is more suitable for embedded purposes. The hardware system is usually implemented on an Application Specific Integrated Circuit (ASIC) or Field Programmable Gate Arrays (FPGAs). For use in testing or prototyping, FPGAs are widely used instead of ASICs. Implementation on an FPGA is useful
Figure 1. Example of Haar-like features. These simple features are reminiscent of Haar basis function.

for computer vision, as shown in [3, 7]. Although examples of specialized hardware for object detection have been proposed in [5, 8, 12, 13], no system achieves both a high detection rate and fast processing.

In this paper, we propose a robust and rapid detection system with a specialized processor suitable for object detection. To realize real-time processing, our architecture features the following two major techniques. One is hybrid parallel execution, which exploits the non-uniformity of computation cost in the detection algorithm. We assign more resources to frequently executed tasks than those less frequently executed. The other is a method of scaling input images, which simplifies hardware implementation while maintaining a high detection rate. In addition, we implement the architecture on a commercially available FPGA to show that our system performs real-time detection on higher resolution images or at a higher frame rate than existing hardware.

2. Related work

The first half of this section describes an object detection algorithm based on AdaBoost learner using Haar-like features, and the last half introduces existing hardware for object detection.

2.1. Object detection based on AdaBoost

Viola and Jones [9] proposed a rapid and robust object detection method by using a variant of the AdaBoost algorithm both to select the features and to train the classifier. AdaBoost is one supervised learning scheme employed to boost the classification performance of a simple learning algorithm. In their scheme, a feature extraction method called ‘Haar-like feature’ is used for weak classifiers. Haar-like features resemble Figure 1, which can detect an edge or a line feature and resembles the Haar basis function. Haar-like features are obtained from input images by the following operations: first the sum of the pixel values within the white and black regions in Figure 1 are calculated respectively, and then the difference of the weighted sum of these regions are calculated. If the difference exceeds a predefined threshold, the weak classifier outputs true, which means an edge feature exists. If not, the weak classifier outputs false. By combining these weak classifiers, a boosted strong classifier that can detect more complicated objects is realized. Note that the computational cost to calculate the sum of the pixel values in a rectangle is reduced using integral images.

Detection is performed on each rectangle region called a sub-window, which scans the entire input image. After all sub-windows in one frame are evaluated, similar procedures are iterated for another sub-window size to detect objects of different sizes.

To realize rapid detection, Viola and Jones proposed a cascade structure of a classifier that is a series of stage classifiers (see Figure 2). This cascade structure drastically reduces processing time because the classifier is trained to quickly reject non-object sub-windows and spend more computation on promising object-like regions.

2.2. Existing hardware for object detection

Here, previous works about hardware implementation of object detection are introduced.

Theocharides et al. [8] implemented a face detection system on an ASIC to accelerate processing speed. To exploit parallelism, they use a grid array processor as the structure of their architecture. This enables high-speed calculation of classifiers and achieves 52 frames / second (fps). However, a large processor array which is identical to the input image size is required and it may increase when the image size becomes larger. Image size and detection rate are not discussed in [8]. Wei et al. [12] chose an FPGA instead of an ASIC as a target device. They use techniques such as scaling input images and fixed-point expression to achieve fast processing with less circuit area. The architecture is implemented on a Xilinx® Virtex™II FPGA and achieves 15 fps at 91 MHz. However the image size is 120 × 120, and only some parts of a classifier cascade are actually implemented. Another implementation of a face detection system on an FPGA is reported by Yang et al. [13]. To realize a low-cost detection system, the architecture is implemented
on an inexpensive ALTERA® CycloneTMII FPGA. Their architecture achieves 13 fps. On the other side of low-cost implementation, the detection rate falls to about 75% while in the original method by Viola and Jones [9] it is more than 90%. On the other hand, Nair et al. [5] proposed an embedded system for human detection. This is also implemented on an FPGA, which performs on an input image of about 300 pixels but the frame rate is only 2.5 fps.

As the above examples, object detection hardware that achieves both a high detection rate and little processing time has not been proposed yet. Therefore we propose a hardware architecture for AdaBoost-based object detection that can maintain a high detection rate and achieve a high frame rate on high resolution images.

3. Ideas for hardware implementation

As described in Section 1, fast processing is required to achieve real-time object detection. Sequential execution on a high frequency processor or a multiprocessor is one of the popular fast methods, but this is not suitable for embedded use due to its high power consumption or large size of an entire system. Therefore, in this section we focus on parallel execution and discuss some effective methods for rapid execution. In addition, we propose a method of scaling input images suitable for hardware implementation.

3.1. Parallelism in a detection algorithm

Available parallelism in the AdaBoost-based detection algorithm is as follows (also shown in Figure 3):

1. Scale factors of a sub-window: In this detection algorithm, the scale factor of the sub-window size takes some variations (usually approximately 10 factors). The calculation of each scale factor can be performed independently.

2. Partitioned input images: Large input images can be separated into several partitions. Processing on each partitioned image can be executed in parallel since such images are independent of each other. Note that image partition does not work when the scale factor is too large (see the top example of the scale factor in Figure 3).

3. Weak classifiers: The strong classifier in the AdaBoost algorithm contains a large amount of weak classifiers, which can be computed in parallel. The number of weak classifiers is approximately several thousands, extremely high parallelism can be obtained.

4. Arithmetic and logic operations: Almost all applications have operations that can be executed in parallel.

The parallelism is roughly classified into two groups: coarse-grained and fine-grained. The parallelism of scale factors or partitioned images belongs to coarse-grained, and the parallelism of weak classifiers or operations belongs to fine-grained. Parallel systems with multiprocessors or multi-core processors are suitable for exploiting coarse-grained parallelism. Fine-grained execution on such processors may cause large overheads. On the other hand, the embedded hardware is not only suitable for coarse-grained parallel execution but also for fine-grained. Therefore, all the parallelism described above is available for hardware implementation.

In this paper, we focus on fine-grained parallelism to realize the proposed specialized processor for the following two reasons: One, coarse-grained parallelism is not enough to achieve real-time processing on high resolution images or at a high frame rate. Two concerns scalability for future extension. By using a number of proposed processors, we can also exploit coarse-grained parallelism to perform detection on larger images or at a higher frame rate.

3.2. Parallel execution for embedded hardware

As discussed above, we propose a specialized processor that exploits fine-grained parallelism. This part describes a method of parallel execution for weak classifiers and its effect on processing time.

3.2.1 Parallel execution of weak classifiers

A trained classifier consists of thousands of weak classifiers that calculate Haar-like features from the integral images, as described in Section 2. These weak classifiers can be computed in parallel. Figure 4 shows a concept of the architecture for parallel computing of the weak classifiers. Each
Figure 4. Parallel execution for weak classifiers. Each weak classifier module has a Haar-like feature evaluated in $P$ parallel.

module for the weak classifier works independently.

With this parallel execution, processing time is ideally diminished to $1/P$ if the $P$ modules work simultaneously. However, the processing speed does not always become $P$ times faster because the effect of parallel execution differs between the former and subsequent stages in the cascade. Since the classifier cascade is trained to reject most non-object sub-windows at the early stage, the former stages are executed frequently but not the subsequent stages. Considering this fact, parallel execution on the former stages might be effective for rapid processing. This will be discussed in the next part.

3.2.2 Partially parallel approach

The execution of all the thousands of weak classifiers in parallel is the fastest method. However, this is not efficient because the subsequent stages are rarely executed, even though the former stages are almost always executed. Therefore parallel execution of only the former stages is reasonable to accelerate the total processing performance. A problem remains: how many former stages should be executed in parallel to achieve enough performance? To determine this, we examine the distribution of the execution rate for each stage by software profiling and discuss the effect of the parallel execution of the former stages.

We use the detection program based on OpenCV Library [1] for software profiling. The target objects are frontal human faces, according to Viola and Jones [10]. We use trained data provided by OpenCV, which is trained by frontal faces whose size is $24 \times 24$, that includes a total of 25 stages and 2913 Haar-like features. Table 1 shows the number of weak classifiers in each stage. Note that the former stages include fewer weak classifiers than the subsequent stages. Detection is performed on a MIT+CMU test set [6] that consists of 130 images with 507 labeled frontal faces. The scale factor is set to 1.2, and the scanning step is set to 1.

![Sub-window](image)

**Table 1.** Number of weak classifiers in each stage

<table>
<thead>
<tr>
<th>stage number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak classifiers</td>
<td>9</td>
<td>16</td>
<td>27</td>
<td>32</td>
<td>52</td>
<td>53</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>72</td>
<td>83</td>
<td>91</td>
<td>99</td>
<td>115</td>
<td>127</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>136</td>
<td>137</td>
<td>169</td>
<td>155</td>
<td>169</td>
<td>196</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>181</td>
<td>199</td>
<td>211</td>
<td>200</td>
<td>total</td>
<td>2913</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.** Execution rates of cascade stages. Former stages are executed more frequently than subsequent stages.

**Figure 6.** Processing time rate of cascade stages. Solid line shows rate against total time, and dotted line shows cumulative rate.

The result of profiling is shown in Figure 5. In fact, the former stages are frequently executed but the rate of the subsequent stages falls exponentially. The processing time rate for each stage against the total processing time is calculated from Table 1 and Figure 5. This is shown in Figure 6. The solid line shows the processing time rate. The dotted line shows the cumulative rate, which tells us how much processing time has elapsed in the former stages (e.g., about 85% for stages from 0 to 5 and about 96% for 0 to 10). This means parallel execution is very effective only on the former $N$ stages to decrease the total processing time. With parallel execution of the former stages, calculation cost for the subsequent stages becomes slight enough to be executed sequentially. When applying this approach to a real sys-
tem, the number of former stages should be determined in consideration of a tradeoff for circuit area and processing performance.

### 3.3. Scaling images

In Viola and Jones’s [10] object detection algorithm, sub-windows are expanded to detect large objects, which is not difficult because the Haar-like feature consists of simple rectangle features that are easy to enlarge. Since scaling sub-windows does not need processing of the input images, this method is widely used for software implementation [2, 10]. When the sub-window size becomes larger, however, data access to the integral image becomes sparser because the size of the rectangles in the Haar-like feature also increases. This may frequently cause a cache miss in the case of CPUs or require large cache memory for a specialized processor to achieve fast memory access.

On the other hand, one method scales input images instead of the sub-windows. A scaling image technique is used in existing hardware [12, 13] because it is suitable for hardware implementation for the following reasons: First, as described above, there is no need to prepare a huge cache memory for rapid memory access. Next, it is easy to implement on hardware because once the input images are scaled, the subsequent process is completely identical at every scale factor. Finally, processing time overhead for generating the scaled image can be concealed by pipelined implementation. Note that caution must be exercised because image scaling might affect the detection rate. This topic will be discussed in the next section.

### 3.4. Section summary

First, this section describes the parallel execution methods for rapid object detection. Among them, parallel implementation of weak classifiers is found suitable for embedded hardware. Next, the partially parallel implementation of the former stages is shown to increase the total processing performance, since former stages are used more frequently than subsequent stages. Finally, a technique of scaling images instead of enlarging sub-windows is introduced as another fast method for hardware implementation.

### 4. Evaluation of detection rate

The method of scaling images may affect the detection rate, as described in Section 3. In this section, we evaluate the detection rate of the scaling image method and compare it to the existing method. The other topic in this section is fixed-point implementation, which improves the hardware performance but may lower the detection rate due to a lack of accuracy. We compare the detection results of floating-point and fixed-point implementations to explore moderate bit width for fixed-point variables.

#### 4.1. Detection rate of scaling image method

In this subsection, the detection rate of the scaling image method is compared to the existing scaling sub-window method. The experiments are conducted under the same condition as Section 3 using the OpenCV library. The scale factor is also set to 1.2, which is also identical to Section 3.

To make small images, we test two major scaling algorithms: nearest neighbor interpolation and bilinear interpolation. In the nearest neighbor interpolation algorithm, a pixel value in the new image is set to the value of the nearest pixel in the original image. This is the simplest interpolation algorithm that requires lower computation cost. On the other hand, the bilinear interpolation algorithm can generate better scaled images than the nearest neighbor method, but it requires more computational cost. In this algorithm, a pixel value in the new image is calculated from the values of the four nearest pixels (up, down, left, and right) in the input image.

Figure 7 shows detection results with Receiver Operating Characteristic (ROC) curves of the three methods: nearest neighbor, bilinear, and existing scaling sub-window. ROC curves show the relationship between both the detection and false positive rates. The detection rate is the number of correctly detected objects against the number of true objects (507), and the false positive rate is the number of inaccurately detected sub-windows against all evaluated sub-windows (about 81 million). To create ROC curves, the threshold for the grouping of detected sub-windows is adjusted. Adjusting the threshold to a low value will yield high detection and high false positive rates. Adjusting the threshold to high, however, lowers both the detection and false positive rates.

In Figure 7, the solid line denotes the result of the existing method of scaling sub-windows, the dotted line denotes the result of the scaling image method with nearest
neighbor interpolation, and the broken line is for bilinear interpolation. The nearest neighbor method shows better performance than the existing method in the high detection rate region and identical performance in the low detection rate region. The bilinear method shows better performance in the mid-range of the detection rate and the same as the other methods in the other region. These facts show that the performance of the method of scaling images is almost the same as the existing method. It is also shown that the nearest neighbor interpolation is good for the high detection rate region, and the bilinear is good for the low detection rate region. In this paper we adopt the nearest neighbor method due to its low computational cost.

4.2. Detection rate of fixed-point implementation

For implementation on embedded hardware, fixed-point representation of fraction contributes faster processing and less circuit area than floating-point representation. In this subsection, we examine the effects on the detection rate of fixed-point implementation.

Four variables of fraction affect the detection rate in the detection algorithm:

1. Variance of pixel values in a sub-window
2. Threshold for a weak classifier
3. Weight for a weak classifier
4. Threshold for a cascade stage

The image in the sub-window must be normalized by the variance of pixel values because the detector is trained by the normalized example images. Since the variance values for 8-bit depth images can be expressed in 8-bit, we set the bit width of the variance to 8-bit.

The thresholds for weak classifiers directly affect the detection performance. Figure 8 shows the results of the detection and false positive rates while adjusting the bit width of the threshold. The conditions of the experiment are identical as previous sections. The detection rates are nearly constant but slight decrease is found in the less than 12-bit region. The false positive rates are also constant in the over 13-bit region, but begin to increase in the less than 13-bit region. Thus, we adopt 13-bit for the weak classifier threshold.

The bit widths of the weights for the weak classifiers and the thresholds for the cascade stages are related to each other. Based on the classification results of the weak classifiers in a stage, the predefined weights are accumulated to the stage sum. Since the sum is compared with the stage threshold, the weight and the threshold must have identical accuracy. Therefore we simultaneously examine these variables. Figure 9 shows the results of the detection and false positive rates while adjusting the bit width of the stage threshold. Only the threshold is shown in the figure for simplicity. The detection rates are nearly constant in the over 8-bit region, but they begin to vary in the less than 8-bit region. The false positive rates are also unstable in the less than 8-bit region. Therefore we set the bit width of the weight and the threshold to be 8-bit.

In summary, the minimum bit widths sufficient to main-
tain detection performance are 8-bit for the variance, 13-bit for the threshold of the weak classifiers, and 8-bit for the weak classifier weights and stage thresholds. To compare with the floating-point expression, Figure 10 shows three ROC curves, which are the results of the existing scaling sub-window method, the proposed scaling image method with floating-point, and one with fixed-point. Although fixed-point implementation is slightly worse than floating-point, it achieves almost equal detection performance to the existing method. This shows that fixed-point implementation can maintain high detection performance.

5. Proposed architecture

The proposed architecture for object detection is described in this section. Implementation results on a FPGA are also shown in the last part of this section.

5.1. Architecture overview

Section 3 shows that parallel execution on the former stages in the classifier cascade leads to remarkable reduction of processing time. Therefore, we propose a hybrid architecture of the parallel processing module for the former stages and a sequential processing module for the subsequent ones. This approach achieves high processing performance and realizes smaller circuit area than parallel implementation of all weak classifiers. Figure 11 shows an overview of the proposed architecture.

In the parallel processing module for the former stages, first an input image, which is rotated 90 degrees to save line buffer size, is loaded from the frame buffer, next the image is scaled down based on a current scale factor, and then integral and squared integral images are generated. Note that squared integral images are used to calculate the variances to normalize the sub-windows. The integral images are stored in the line buffers to reuse the data by the neighboring sub-windows. Then the sub-window pixels are transferred to a $25 \times 25$ array of registers from the line buffers. The size of the array is larger than the size of the sub-windows ($24 \times 24$) because the bottom and the right edge pixels are needed to calculate the Haar-like features from the integral image. The register array is connected to many of the modules of the weak classifiers in the former stages. This processing flow—scaling images, generating integral images, and computing weak classifiers—is all pipelined.

The sequential processing module for subsequent stages receives a location of the sub-window accepted by the former stages and sequentially computes the rest of the weak classifiers. Similar to the former module, the subsequent module also performs scaling images, generating integral images, and transfers them to a register array. Then the Haar-like features are calculated in the sequential processor.

5.2. Decision of stage partition

In this subsection we discuss how many former stages should be executed in parallel to realize real-time object detection with our proposed architecture.

The number of sub-windows in one frame is calculated from the size of the input image and the scale factor, and the number of sub-windows per second is also calculated from them. Let the total sub-windows per second be $T$, and a clock frequency of the target device be $F$. The processing on a sub-window must be finished in $C = \left\lfloor \frac{F}{T} \right\rfloor$ cycles on average. Note that $\lfloor x \rfloor$ is the floor function for $x$. Parallel processing on the former stages can easily be finished in $C$ cycles by adjusting the parallel number of weak classifiers that are simultaneously calculated. Therefore the number of former stages should be large enough to finish the subsequent stage processing in $C$ cycles.

The module for the subsequent stage requires startup cycles to generate the integral image in the sub-window. Execution of this process is impossible in advance because subsequent stages do not know which sub-window will be accepted by the former stages. Therefore the modules have to generate integral images after informed which sub-window has passed the former stages. In subsequent stages, it takes $S + A_N$ cycles for a sub-window on average, where $S$ is the startup cycles and $A_N$ is the average processing cycles for computing weak classifiers in the subsequent stages on and after $N$. With the above discussion, the following condition can be satisfied to achieve real-time execution:

$$R_N(625 + A_N) < \left\lfloor \frac{F}{T} \right\rfloor,$$

where $R_N$ is the execution rate of stage $N$ shown in Figure 5 in Section 3.

In this paper, an input video format is assumed to be VGA size $(640 \times 480)$ and 30 fps, and the scaling factor is set to 1.2, as in Sections 3 and 4. The number of
sub-windows that should be evaluated per frame is calculated as 928,145 under the above condition. Accordingly the number of sub-windows per cycle is calculated as \( T = 27,844,350 \). If \( F \) is 140 MHz, \( C \) is calculated as 5. Startup cycles \( S \) are the same as the sub-window size of \( 25 \times 25 = 625 \) in our implementation. Thus, we split the cascade stages at \( N = 10 \) to satisfy Equation (1), where \( R_{10} \approx 0.45\% \) and \( A_{10} \approx 449 \) lead \( R_N(625 + A_N) \approx 4.8 \), which is less than \( C = 5 \). With \( N = 10 \), the former module computes about 100 weak classifiers per cycle (497 weak classifiers per 5 cycles), and the subsequent module does one per cycle.

5.3. FPGA implementation

The proposed architecture is implemented on Xilinx\textsuperscript{®} Virtex\textsuperscript{™}-5 FPGA (XC5VLX330-2) to evaluate performance and area utilization. The synthesis results are shown in Table 2. Block RAMs means embedded memories on the FPGA, and DPS48Es means DSP blocks that include embedded multipliers. The frame and line buffers are implemented using Block RAMs. The maximum frequency of this implementation is 160 MHz.

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>68145</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>74615</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>152</td>
</tr>
<tr>
<td>DPS48Es</td>
<td>112</td>
</tr>
</tbody>
</table>

Table 2. Synthesis results of implementation on a Virtex-5 FPGA (XC5VLX330-2). Block RAMs means embedded memories on the FPGA, and DPS48Es means DSP blocks that include embedded multipliers.

6. Conclusion

In this paper, we proposed a specialized processor suitable for real-time object detection based on the AdaBoost algorithm. To perform rapid detection, we adopt hybrid implementation of parallel processing and sequential modules. Frequently used stages are executed in parallel, and subsequent stages are executed sequentially. This drastically improves the total processing time without a great increase of circuit area. In addition, the techniques of scaling input images and fixed-point implementation are also used in our architecture. We demonstrate that the high detection rate performance of these methods is identical to existing software implementation. Finally the proposed architecture is implemented on a Virtex\textsuperscript{™}-5 FPGA. We realize both the parallel and sequential parts as embedded hardware, but the sequential processing can be performed on CPUs in practical use. The results show our architecture achieves real-time detection on VGA video at 30 fps.

Acknowledgements

This work is supported by the VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Mentor Graphics, Inc.

References