An Implementation of Embedded Object Detection System with Information-Preserved Algorithm Transformation

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2019

Koichi MITSUNARI

Publications

Transactions

1. K. Mitsunari, Y. Takeuchi, M. Imai, and J. Yu, "Decomposed vector histograms of oriented gradients for efficient hardware implementation," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101-A, no. 11, pp. 1766–1775, Nov. 2018.

2. K. Mitsunari, J. Yu, T. Onoye, and M. Hashimoto, "Hardware architecture for high-speed object detection using decision tree ensemble," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101-A, no. 9, pp. 1298–1307, Sep. 2018.

International Conference Papers (Refereed)

1. K. Mitsunari, J. Yu, and M. Hashimoto, "Hardware architecture for fast general object detection using aggregated channel features," in *Proceedings of IEEE Asian Solid-State Circuits Conference*, Nov. 2018, pp. 55–58.

2. E. Aliwarga, K. Mitsunari, J. Yu, T. Onoye, T. Azuma, and M. Koga, "System design of vision-based framework for senior driver assistance," in *Proceedings of Workshop on Synthesis And System Integration of Mixed Information Technologies*, Oct. 2016, pp. 77–80.

3. K. Mitsunari and J. Yu, "Influence of numerical precision on machine learning and embedded systems," in *Proceedings of International Workshop on Smart Info-Media Systems in Asia*, Sep. 2016, pp. 164–169.

4. K. Mitsunari, J. Yu, Y. Takeuchi, and M. Imai, "Object tracking based on path similarity of boosted decision trees," in *Proceedings of International Technical Conference on Circuits/Systems, Computers and Communications*, Jul. 2016, pp. 563–566.

Summary

Demand for advanced driver assistance systems (ADAS) based on visual object detection is increasing for reducing deaths and economic loss due to traffic accidents. These embedded systems require fast and accurate object detection with limited power consumption. Due to the severe constraint on power consumption, hardware-oriented design optimization is a promising approach. However, it is still difficult to satisfy the above requirements simultaneously because many existing object detection algorithms are not designed considering hardware implementation. To address this issue, this dissertation proposes a hardware architecture for an object detection method with aggregated channel features (ACF). This dissertation approaches the issue through information-preserved algorithm and hardware architecture for fast classification. For improving the trade-offs, this dissertation focuses on information preservation in histograms of oriented gradients (HOG) feature descriptor and a quantization method for boosted decision trees (BDT) classifiers, and highly-parallelized hardware architecture for BDT.

For improving the trade-off between detection accuracy and power consumption of feature extraction, this dissertation proposes information-preserved HOG feature descriptor named decomposed vector HOG (DV-HOG). DV-HOG feature extraction is based on the decomposition of a gradient vector to generate a histogram. DV-HOG extracts equivalent information to the original HOG, and it can be computed only with additions and multiplications. The hardware architecture for DV-HOG utilizes the symmetry of the vectors to reduce power consumption. Experimental results show that DV-HOG achieves the equivalent or better detection accuracy to the original HOG only with one-fourteenth hardware area.

For reducing memory requirements, this dissertation proposes a quantization method for a BDT classifier, which ACF uses as a classifier. The proposed method utilizes the BDT's characteristics that BDT is based on the comparison of a pair of a feature and a threshold. Thus, the range of thresholds of a BDT classifier is narrower than that of features. The proposed quantization method focuses only on the range of thresholds for quantization. Experimental results show that the memory requirement can be reduced to one-sixteenth with 2% accuracy degradation on INRIA Person Dataset, which improves the trade-off between the detection accuracy and memory requirement.

For improving detection speed, this dissertation proposes a hardware architecture for fast BDT classification. The fast classification is realized by hardware-software cooperative approach: highly-parallelized hardware and a software algorithm for avoiding memory access conflict from multiple hardware modules. The hardware supports 3-D parallelized classification: 2-D for images like SIMD operation and 1-D for feature channels. The hardware is designed to reduce hardware resources and connections for improving the scalability to the high degree of parallelism. The scheduling algorithm using a greedy approach determines the memory access pattern before classification. The evaluation result shows that 1,024-parallel implementation is capable of classifying pedestrian in 350 frames of Full HD images.

Based on the above three methods, this dissertation proposes an ACF object detection hardware. Thanks to the high compatibility of ACF to hardware implementation and the above hardware-oriented algorithms to overcome the challenges, the hardware achieves fast and accurate classification. An FPGA implementation result shows that the proposed system can detect pedestrians in 170 fps a Full HD image, which is 57-times faster than the existing ACF hardware implementation. As an evaluation in a practical environment, the proposed hardware can process 6-class traffic object detection in 78 fps for a Full HD image, which satisfies the requirement for the automatic braking system of ADAS.

The main contribution of this work is the improvement of trade-offs between detection accuracy, detection speed, and power consumption in object detection, which is achieved by the use of the information-preserved algorithm and hardware-oriented approaches.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor Professor Masanori Hashimoto in Osaka University for his insightful comments and valuable advice. He carefully reviewed my papers and gave me much advice for completing this dissertation.

I would like to acknowledge Professor Takao Onoye in Osaka University for his insightful comments to this dissertation. Also, he provided me with precious opportunities to attend many research projects during my graduate studies. I would like to express my appreciation to Professor Haruo Takemura in Osaka University for reviewing my dissertation and giving me valuable suggestions to raise the quality of this dissertation as a member of my dissertation committee.

My appreciation for Assistant Professor Jaehoon Yu in Osaka University cannot be expressed in words. Without his proper guidance and continuous encouragement, this research would not have been possible.

I deeply appreciate Emeritus Professor Masaharu Imai in Osaka University and Professor Yoshinori Takeuchi in Kindai University for providing me with opportunities to study at Imai Laboratory while I was pursuing master's degree.

I would like to thank all the members of the Integrated System Design Laboratory in Osaka University for daily discussion and having a comfortable time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Health Organization reported in 2018 that more than 1.25 million people died due to traffic crashes each year, and these accidents made 3% of gross domestic product loss for most countries [1]. In Japan, according to the National Police Agency, there exist 3,694 deaths caused by traffic accidents [2], and 89% of the accidents are collisions [3]. For saving human lives and reducing economic loss, preventing traffic accidents is demanded as a social requirement. Of the entire traffic accidents, there exist some physically inevitable cases, but the majority originates from drivers' human errors such as operation misses and distracted driving, which is preventable with the advanced sensor technology and machine learning techniques. Advanced driver assistance system (ADAS) is a representative system developed for this purpose and provides driving assistant functionalities including collision avoidance system, driver monitoring system, automotive navigation system, etc.

Of many components composing ADAS, the collision avoidance system plays an important role in preventing traffic crashes by controlling the braking system. Figure 1.1 shows the stopping distances of human drivers at various car speeds [4], where the stopping distance is a sum of reaction distance needed for driver's response and braking distance needed for deceleration to still stand. If the vehicle speed is 60 km/h, which is the speed limit of urban areas in Japan, the stopping distance is 45 meters in which 25 meters is the reaction distance. Collision avoidance system assesses collision risks of objects on the road instead of drivers and can reduce the reaction distance significantly. Although range sensors such as millimeter-wave radar provide enough information for the basic braking functionality, recent ADAS requires a more comprehensive assessment of the traffic environment, and as a result, the demand for visual object detection is

[1] http://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[2] https://www.npa.go.jp/toukei/koutuu48/H29siboubunnseki.pdf

[3] https://www.e-stat.go.jp/stat-search/file-download?statInfId=000031674176&fileKind=2

[4] https://www.qld.gov.au/transport/safety/road-safety/driving-safely/stopping-distances

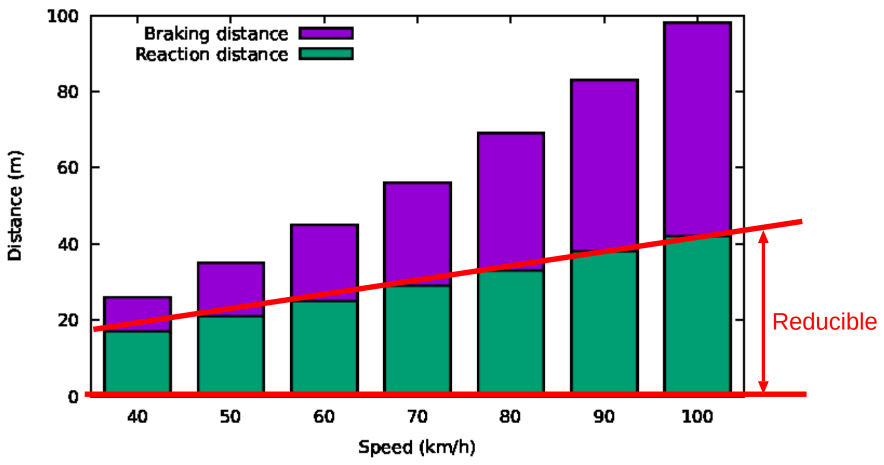


Figure 1.1: Stopping distances for various running speed.

increasing.

Visual object detection for embedded systems needs to satisfy the following requirements:

1. high detection accuracy,
2. real-time responsiveness, and
3. low power consumption.

These requirements are more stringent in ADAS than other practical applications such as surveillance systems and autonomous robots because high detection accuracy and real-time responsiveness are indispensable to guarantee the safety of ADAS. However, it is difficult to satisfy these requirements simultaneously under the constraint of low power consumption. In the case of early object detection algorithms [1–4], they require low computational cost while it is difficult to achieve high detection accuracy. Therefore, their hardware implementations operate with low power consumption but their accuracy was confined to algorithmic limitation [5–13]. On the other hand, recent object detection algorithms provide high enough detection accuracy for practical applications [14–22]. These methods require large computational cost, but thanks to advanced parallel computing devices such as GPUs, it is possible to process them in real time. A critical problem is that these devices consume more than a hundred watts, which is not affordable for the embedded systems. Dedicated hardware implementation makes it possible to reduce this power consumption, but it still does not satisfy both processing performance and power consumption for practical applications [23]. To improve the trade-off relationship between three requirements, this dissertation focuses on ADAS and will propose an object detection system focusing on information preservation,

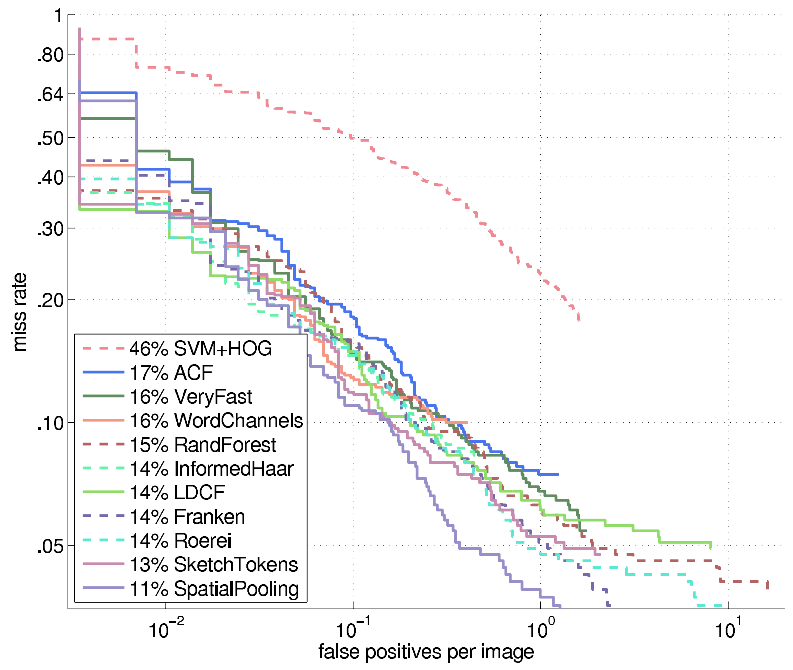


Figure 1.2: Detection error trade-off (DET) curves of leading-edge algorithms and SVM+HOG [24] from pedestrian detection benchmark [25]. The closer a curve gets to the bottom left corner, the better the DET is.

which enables both high detection accuracy and real-time responsiveness with low power consumption.

1.1 Background on Object Detection

Visual object detection has shown remarkable progress regarding both processing performance and detection accuracy in the last decade [25–28], thanks to the sophisticated machine learning algorithms [14, 15, 29, 30] and feature descriptors [24, 31–35]. As shown in Figure 1.2, novel object detection methods such as aggregated

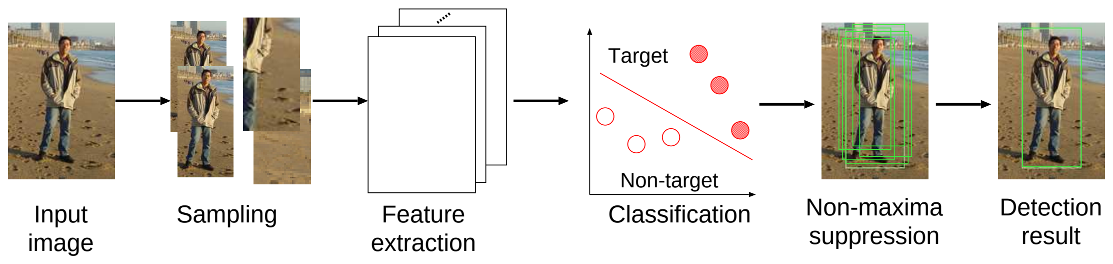


Figure 1.3: Object detection flow of conventional approaches.

Table 1.1: Comparison of existing object detection hardware implmentations.

Implmenetation	[13]	[23]	[40]
Method	SVM	CNN	ACF
Log-average MR on INRIA Person Dataset [24]	46%	<10%	17%
Speed	Full HD 60 fps	1.3 fps	VGA 30 fps

channel features (ACF) [36] and VeryFast [37] achieve about 30% lower log-average miss rate (MR) than SVM+HOG [24] on INRIA Person Dataset [24]. Especially, the state-of-the-art methods based on deep neural networks (DNNs) [14–22] succeeded in rich representations of objects through a massive amount of computation with the help of powerful computational resources such as GPUs and achieved high detection accuracy on public datasets such as ImageNet [38] and CityPersons [39].

As for hardware implementation, Table 1.1 compares the three existing implementations [13, 23, 40] in terms of detection accuracy and speed, where each implementation is based on support vector machine (SVM), convolutional neural network (CNN), and ACF, respectively. The implementation using SVM [13] achieves detection fast enough for ADAS, but its detection accuracy is low due to the limitation of the classification capability of a linear classifier and is not acceptable for ADAS. On the other hand, the implementation using CNN [23] achieves the state-of-the-art detection accuracy, but the detection speed is not enough for ADAS. Since CNN is based on MAC operations, there exist inherent limitations of speed-up and resource reduction by hardware implementation. Although approaches that reduce numerical precision such as binary and ternary networks [41, 42] are proposed, it is difficult to ensure high detection accuracy for the state-of-the-art networks. ACF is an object detection method using aggregated channel features and decision tree based classifiers. The ACF hardware implementation [40] shows sufficient detection accuracy and moderate processing performance compared with two other methods. Although the detection accuracy is somewhat inferior to that of CNN, recent research such as deep forest [43] shows that detection accuracy of decision tree based classifiers can be improved by deep learning. This approach is expected to improve the detection accuracy of ACF.

Figure 1.3 summarizes a typical object detection flow using a sliding window based approach. It mainly consists of three steps: sampling, feature extraction, and classification. First, detection windows are sampled from an input image across image positions and scales. Next, features are extracted for each window. The features include, for example, color, and appearance information. Feature extraction converts the input image to the feature space, which helps the following classification step. Then, classifiers separate the feature space into two subspaces: a target class and the others. Classification algorithm influences the detection accuracy because the representation capability is different among machine learning methods, which will be explained in

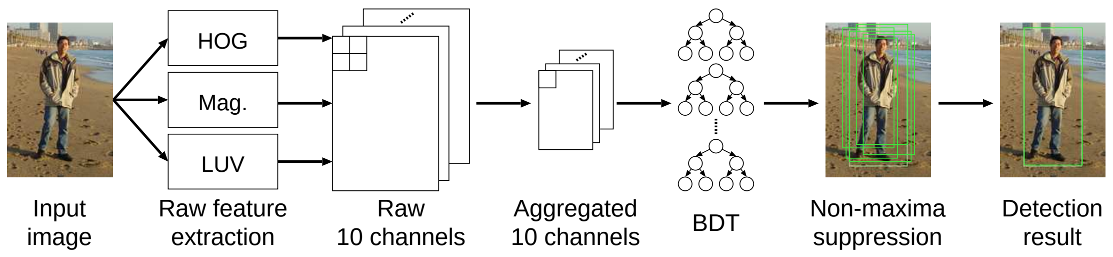
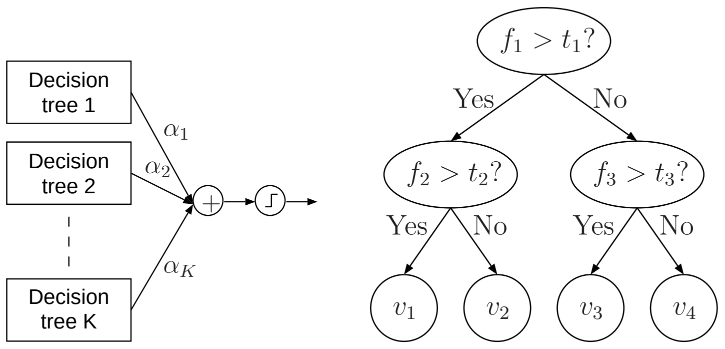


Figure 1.4: ACF object detection flow.



(a) Boosting for classification. (b) Decision tree.

Figure 1.5: Boosted decision trees.

Chapters 2 and 4.

This dissertation focuses on the ACF object detection method [36] as the target for hardware implementation because of the balance between detection accuracy and speed mentioned above. Its small amount of computations and small memory requirement is also advantageous in hardware implementation. Dollár *et al.* reported in [36] that a software implementation on a single CPU can process 31.9 fps for a VGA image and achieves reasonably high accuracy on a public dataset. Regarding the balance of the detection accuracy and processing speed, using ACF algorithm is a promising approach for ADAS.

Figure 1.4 shows the object detection flow of ACF. ACF uses HOG feature, gradient magnitude, and LUV color channels as raw features, where the HOG feature represents objects' shape information. After the raw feature extraction, ACF accumulates adjacent feature values of each channel to generate aggregated channels. Then, ACF uses boosted decision trees (BDT) for classification as shown in Figure 1.5. A BDT classifier consists of multiple decision trees (DTs) as in Figure 1.5(a), and each decision tree is shown in Figure 1.5(b). A BDT classifier is trained by boosting algorithm such as AdaBoost [30]. Finally, as post-processing, non-maxima suppression (NMS) merges multiple detection results that correspond to one object.

ACF has high compatibility with the hardware implementation regarding the following three points: small memory requirement thanks to channel aggregation, small hardware resources requirement in classification, and compatibility with a conventional speed-up method for classification. In feature extraction, channel aggregation by 4x4 reduces the required memory capacity to one-sixteenth. The necessary hardware resource for the classification is small because the BDT classifier is based on successive comparisons of pairs of a feature and a threshold from its root node to a leaf node as shown in Figures 1.5(a) and 1.5(b), and does not require multipliers. Also, BDT can use soft cascade [44], which rejects negative samples early, for fast detection since BDT involves a cascade structure. Even with the advantages of hardware implementation, there exists little research on the hardware implementation of ACF. The following points out problems to be addressed for high-throughput ACF implementation.

1.2 Challenges

It is difficult to satisfy the requirements of fast and high detection accuracy with low power consumption in object detection hardware. Most research on hardware implementations [5–8, 11–13, 45–51] remains focusing on classical object detection methods, such as SVMs [24, 32], and shallow neural networks (NNs). However, hardware implementation using SVMs and shallow NNs suffer from poor detection accuracy and cannot fully exploit the algorithmic improvements mentioned in Section 1.1 with Figure 1.2. On the other hand, accurate detection with state-of-the-art DNN algorithms with rich object representations generally demands massive computations, which prevents fast detection and low power implementation. Thus, recently, hardware architectures for DNNs often use limited numerical precision computations [52], which degrades detection accuracy in exchange for detection speed-up.

To address this issue, this research focuses on ACF hardware implementation. ACF achieves reasonably high detection accuracy and ACF uses a BDT classifier, which requires a small amount of computation, and achieves fast detection in software implementation. In the hardware implementation, [40] achieves a classification of 480p30 even with a serial implementation. On the other hand, its parallel hardware implementation pursuing higher throughput is not easy due to the following three difficulties: large hardware resources for the HOG feature extractor, large memory for storing aggregated channel features, and memory access conflict in parallel classification. In terms of parallel classification, the requirements for the large hardware resources make it difficult to increase the parallelism because of limited hardware resources, and the memory access conflict prevents fast classification.

First, the HOG feature descriptor used in ACF requires large hardware resources. HOG is not originally designed for hardware implementation, and its feature extraction contains hardware-unfriendly operations such as square root and trigonometric functions. These operations require large hardware resources for accurate calculation especially in parallel implementation, which makes it difficult to speed-up HOG feature extraction. Thus, existing hardware implementations introduce an approximation for reducing power consumption [8, 11, 48] or permit large hardware resources to ensure no accuracy degradation [5, 6], where the approximation degrades detection accuracy by a few percents [8], and accurate HOG computation occupies as much as 58% of the power of the entire object detection hardware [6].

Second, even with the channel aggregation of ACF, the memory requirement for a high-resolution image is still a large burden to hardware implementation: for a single scale of a Full HD image, 39Mbit memory is required for storing aggregated channel features. Existing hardware implementations use limited numerical precision to reduce memory requirements, but it degrades detection accuracy as a result. This results from the fact that many existing quantization methods focus only on reducing the numerical precision of calculation, but not on exploiting the algorithmic characteristics of the machine learning algorithm.

Finally, classification using BDT machine learning algorithm is difficult to speed-up by the parallel implementation. As explained in Section 1.1, classification using a BDT classifier is based on the comparison of a feature and a threshold, where memory accesses occur for each comparison. Considering the selected paths of a BDT classifier depends on the input data, it is impossible to predict memory access patterns in advance, which causes memory conflict and makes parallel implementation difficult in existing ACF hardware implementation [40].

1.3 Research Objectives

The goal of this research is to build an object detection system satisfying the requirements on detection accuracy, real-time responsiveness, and power consumption. This research focuses on hardware architecture for ACF, whose detection accuracy is reasonably high enough for practical applications. This work improves the trade-offs between detection accuracy, detection speed, and power consumption by handling the challenges explained above.

First, this research focuses on the information-preservation and hardware implementation efficiency for improving the trade-off between detection accuracy and power consumption of HOG feature extraction. Existing hardware implementations require large hardware resources for ensuring no information loss or permit accuracy degradation by using an approximation. In existing hardware implementations, satisfying constraints on both detection accuracy and power consumption are difficult because algorithm modification is limited at the hardware level. This research, on the other hand, approaches in algorithmic level considering hardware implementation efficiency. The proposed feature descriptor is based on a novel HOG feature descriptor using vector decomposition, which has an equivalent representation capability to the original HOG feature descriptor without any information loss.

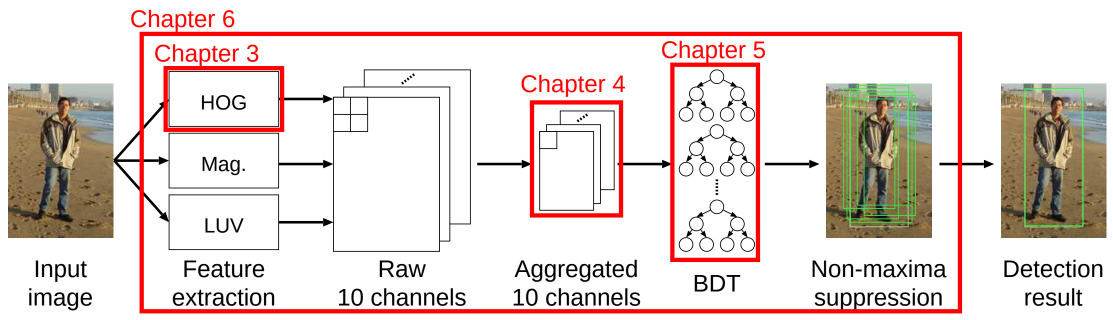


Figure 1.6: Overview of this dissertation.

Second, this dissertation aims to reduce the memory requirements used for storing features in ACF. To tackle the issue, this research focuses on the characteristics of BDT. In many existing hardware implementations using SVMs and NNs, efficient quantization is difficult because they are based on multiplications and only need to represent a wide range of values. Compared with these machine learning algorithms, BDT require a narrow value region since BDT classification is based on the comparison of a pair of values and needs to represent values near the thresholds. Based on this point, this research proposes a quantization method to reduce memory requirements, which focuses on the threshold range of a BDT classifier.

Third, improving the classification speed of BDT is aimed in this dissertation. To address the issue, this research exploits 3-D parallelism of ACF classification: 2-D for images and 1-D for feature channels. As mentioned in the previous section, in parallel processing, BDT suffer from random memory access depending on input data. To prevent memory conflicts, this research adopts SIMD-like operations in a fixed order for parallel processing. Also, this research maximizes memory usage with multiple banks through task scheduling decided in advance by a greedy algorithm. With this hardware-software cooperative approach, significant processing acceleration is achieved.

1.4 Outline of the Dissertation

The rest of this dissertation explains the proposed ACF hardware implementation. Chapters 3, 4, 5, and 6 are the main contributions of this work, which are summarized in Figure 1.6. Table 1.2 shows the challenges and objectives mentioned above. The remaining chapters are organized as follows.

Chapter 2 provides the knowledge required for understanding ACF and its hardware architecture. First, it reviews three types of machine learning algorithms, which are SVM, multi-layer perceptron (MLP), and AdaBoost. Then, it explains the ACF object detection flow and its components, HOG feature descriptor, and BDT classification, followed by the algorithms and hardware implementations of the original HOG and

Table 1.2: Challenges, objectives, and solutions.

	Target	Challenge	Objective	Solution
Chapter 3	HOG	Large power consumption	Computational cost reduction	Decomposed vector HOG
Chapter 4	BDT	High numerical precision	Memory reduction	Quantization focusing on thresholds
Chapter 5	BDT	Slow classification	Fast classification	SIMD-like HW and task scheduling

approximated ones. As the required knowledge for BDT, this chapter explains available hardware architectures for BDT in literature.

Chapter 3 presents the proposed information-preserved HOG feature descriptor, decomposed vector HOG (DV-HOG). Compared with the existing algorithm based on angular interpolation, the proposed DV-HOG uses vector decomposition in the histogram voting method based on linear algebra. This chapter explains its algorithm and shows that the information is preserved in the algorithmic level. Then, a hardware architecture for DV-HOG is presented. The hardware architecture exploits the symmetry of the vector to the x and y-axes to reduce computation and hardware resources. The evaluation shows the detection accuracy and hardware resources of multiple HOG feature descriptors to compare the trade-off between detection accuracy and power consumption.

Chapter 4 presents a quantization method for a BDT classifier. It first explains a preliminary evaluation to show that BDT machine learning algorithm is efficient for implementation with low numerical precision. Exploiting the result that BDT is based on the comparison and it is robust to limited numerical precision, this chapter proposes a quantization method for memory-efficient BDT. The basic idea of the proposed method is that the numerical precision can be reduced as long as comparison result at each decision node does not change. The proposed method quantizes the threshold value focusing on the range of threshold values. The evaluation shows the relationship between numerical precision and required memory for a classifier.

Chapter 5 presents a hardware architecture for parallel computation for a BDT classifier, which is based on hardware-software cooperative approach. The proposed implementation consists of a hardware module and preliminary task scheduling software to control the memory access orders in advance. The proposed highly-parallelized hardware architecture supports SIMD-like processing for both speed-up by increasing the parallelism and maintain easiness by assigning same processing to each component. As a software approach, the task scheduling method avoids memory conflict in parallel classification. This task scheduling problem is an $\mathcal{NP}$-hard problem, and a greedy approach is adopted. An evaluation using pedestrian classifiers shows that the proposed greedy approach contributes to speed-up classification. The required hardware resources and processing time for task scheduling are also presented.

Chapter 6 presents an ACF object detection hardware architecture, which uses the algorithms and hardware architectures proposed in Chapters 3, 4, and 5. This chapter explains the hardware architecture for the ACF object detection in order. First, it explains a hardware module for single-scale object detection followed by the whole hardware architecture of the detection system, which supports multi-class and multi-scale detection. The proposed hardware accelerator is a standalone detection system. An evaluation using FPGA demonstrates the required hardware resources and object detection accuracy in two applications. One is pedestrian detection for comparison with existing implementations, and the other is 6-class traffic object detection taking into account practical usage such as ADAS.

Finally, Chapter 7 summarizes the proposed approaches and the evaluation results. Then, it provides a discussion on the proposed method and explains the future work of this research.

Chapter 2

Preliminaries of Visual Object Detection

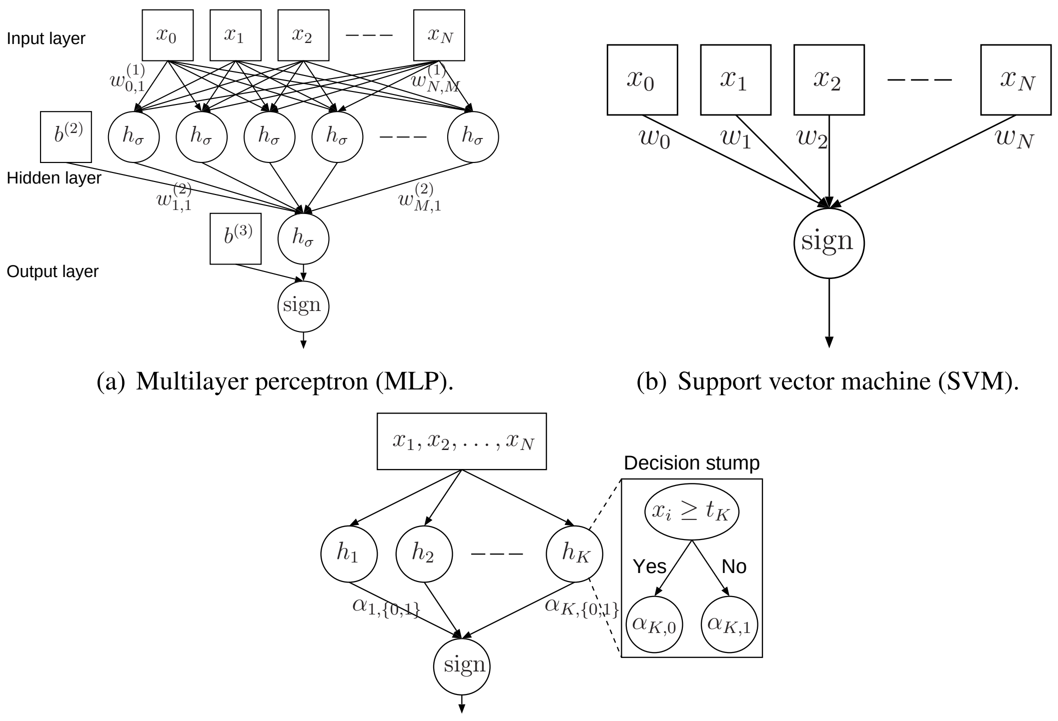
This chapter provides preliminary knowledge required for understanding object detection hardware. First, Section 2.1 describes three major machine learning algorithms, which are used in Chapter 4 for comparison. Next, Section 2.2 explains the ACF object detection method, which is the implementation target of this work. Sections 2.3 and 2.4 explain existing algorithms and hardware architectures of HOG feature descriptor and BDT, respectively, which are the two main components of ACF. Then, Section 2.5 explains pedestrian detection and its evaluation environment, where pedestrian detection is a widely used object detection task used for the evaluation. Finally, Section 2.6 concludes this chapter.

2.1 Machine Learning Algorithms

This section explains three major machine learning algorithms for comparison: an MLP [53], an SVM [29], and an AdaBoost [30] algorithm. Although few recent research uses an MLP algorithm itself, it is used as a comparison target as a representative of convolutional neural networks. This section describes the basic structure of a binary classifier trained by each machine learning algorithm, and how each machine learning algorithm separates the input space.

2.1.1 Multilayer Perceptron

An MLP consists of multiple layers of perceptrons as shown in Figure 2.1(a), where each perceptron calculates an inner product between a weight vector and an input vector. Given a weight vector $\mathbf{w} \in \mathbb{R}^N$ and an input vector $\mathbf{x} \in \mathbb{R}^N$, the perceptron $f(\mathbf{x})$ is defined



(a) Multilayer perceptron (MLP).

(b) Support vector machine (SVM).

(c) AdaBoost.

Figure 2.1: Basic structure of machine learning algorithms.

as

$$f(\mathbf{x}) = h\left(\sum_{i=1}^{N} w_i x_i + b\right), \tag{2.1}$$

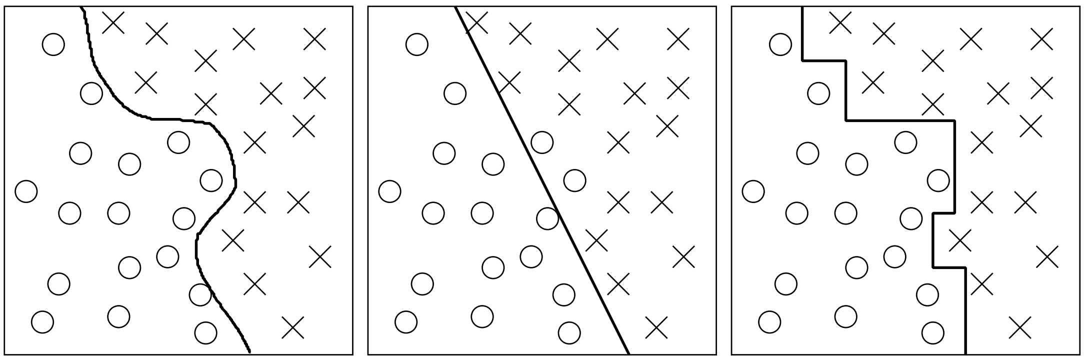
where h is activation function, and b is a bias. Assuming $x_0 = 1$ and $w_0 = b$, the equation can be rewritten as follows:

$$f(\mathbf{x}) = h\left(\mathbf{w}^\top \mathbf{x}\right). \tag{2.2}$$

Based on Eq. (2.2), the MLP shown in Figure 2.1(a) is represented as follows:

$$\begin{aligned} f_{\mathrm{MLP}}(\mathbf{x}) &= \operatorname{sign}\left(f^{(2)}(\mathbf{x}) + b^{(3)}\right), \qquad (2.3)\\ f^{(2)}(\mathbf{x}) &= h_\sigma\left(\sum_{j=1}^{M} w^{(2)}_{j,1} f^{(1)}_j(\mathbf{x}) + b^{(2)}\right),\\ f^{(1)}_j(\mathbf{x}) &= h_\sigma\left(\mathbf{w}^{(1)\top}_j \mathbf{x}\right), \end{aligned}$$

where $\mathbf{w}^{(1)}_j$ is a weight vector from all the input nodes to the j-th hidden node, $w^{(2)}_{j,1}$ is a weight from the j-th hidden node to the output node, $b^{(i)}$ is a bias of the i-th layer, and



(a) Multilayer perceptron (MLP). (b) Support vector machine (SVM). (c) AdaBoost.

Figure 2.2: Input space separation. Each classifier separates two class data: ○ and ×.

the activation function $h_\sigma(x)$ is defined as

$$h_\sigma(x) = 1.7159 \times \frac{1 - \exp(-2x/3)}{1 + \exp(-2x/3)}. \tag{2.4}$$

The final classification result $f_{\mathrm{MLP}}(\mathbf{x})$ is calculated by the following sign function to threshold the prediction value to two classes:

$$\mathrm{sign}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.5}$$

In the training process, backpropagation is used to update the weights and the biases. Figure 2.2(a) shows an example of how an MLP divides the input space in binary classification. Due to the nonlinearity of Eq. (2.3), the representation capability is high, and it is able to separate an input space complexly. The state-of-the-art DNN classifiers contain more layers in the classifier and the number of nonlinear transformations, which is the main reason for the high representation capability of the state-of-the-art DNN classifiers.

2.1.2 Support Vector Machine

The basic structure of an SVM is identical to that of a simple perceptron as shown in Figure 2.1(b), where a linear SVM is discussed for the convenience of explanation. A linear SVM is represented by

$$f_{\mathrm{SVM}}(\mathbf{x}) = \mathrm{sign}\left(\mathbf{w}^\top \mathbf{x}\right), \tag{2.6}$$

where the activation function is the sign function. The only difference between an SVM and a simple perceptron is the training algorithm maximizing the margin between classes in N-dimensional feature space. Figure 2.2(b) shows a classification example. We can see that a linear SVM classifier cannot separate linearly inseparable data, which is one of the most significant reasons of accuracy degradation.

2.1.3 Adaptive Boosting

An AdaBoost is one of ensemble learning algorithms, building a strong classifier consisting of weak learners as shown in Figure 2.1(c), where "strong" and "weak" represent high and low prediction capability, respectively. The strong classifier, which is composed of K weak learners, is represented as follows:

$$f_{\text{AdaBoost}}(\mathbf{x}) = \text{sign}\left(\sum_{k=1}^{K} h_k(\mathbf{x})\right), \tag{2.7}$$

where the k-th weak learner $h_k(\mathbf{x})$ is defined as

$$h_k(\mathbf{x}) = \begin{cases} \alpha_{k,0} & x_i \geq t_k \\ \alpha_{k,1} & \text{otherwise} \end{cases}. \tag{2.8}$$

In the training process, each sample has a weight indicating the difficulty of classifying the sample; the weight is increased when the sample is misclassified and vice versa. This algorithm iteratively selects a weak learner to minimize the total weight of misclassified samples. Figure 2.2(c) shows an example of a classification using AdaBoost. The figure shows that AdaBoost has high representation capability to nonlinear data because of the combinations of threshold values of the classifier.

2.2 Aggregated Channel Features

In computer vision, non-rigid object detection has been a challenging issue and studied for several decades. Of many existing methods, Dollár *et al.* proposed an object detection method named ACF [36], which shows excellent performance in terms of both classification accuracy and computational efficiency. Figure 1.4 shows the processing flow of ACF. As shown in Figure 1.4, the ACF extracts ten raw channels of features from an input image: six channels for HOG [24], a channel for normalized gradient magnitude, and three channels for each of LUV color channels. Then, channel aggregation step accumulates the adjacent features of each memory to compute aggregated channel features. Throughout this dissertation, the block size is set to 4x4 pixels, and the memory requirement of aggregated channels is one-sixteenth size compared with that of raw channels. Each pixel of aggregated channels is looked up by BDT, where depth-two DTs are used as weak learners of the AdaBoost. After classification, multiple detection results are obtained around the target object in general, and these results are clustered by non-maxima suppression.

Since each DT consists of multiple decision stumps, BDT is expected to show a similar result to the AdaBoost explained in the previous section, but in this case, each value of leaf nodes is positive-semidefinite, which means that it is not necessary to consider the sign bit in fixed-point representation. In [36], Dollár *et al.* reported that BDT using ACF achieved 17% log-average MR on INRIA Person Dataset [24] and 31.9 fps processing performance on a single CPU, which outperforms other types of state-of-the-art methods.

2.3 HOG Feature Descriptors and Its Hardware Architectures

The HOG feature descriptor extracts appearance information from images. Many object detection algorithms use this feature descriptor including state-of-the-art methods. For hardware implementation, variants of HOG feature descriptors have been proposed. First, this section explains algorithms for extracting the HOG feature and its variants and then describes hardware architectures for each feature descriptor.

2.3.1 HOG Feature Descriptor

The HOG feature descriptor extracts appearance information from images in the form of a distribution of intensity gradients. Given an input image I, horizontal and vertical intensity gradients are calculated for each pixel. From these two gradients, a two-dimensional gradient vector $\mathbf{g}$ can be defined with magnitude M and orientation θ as follows:

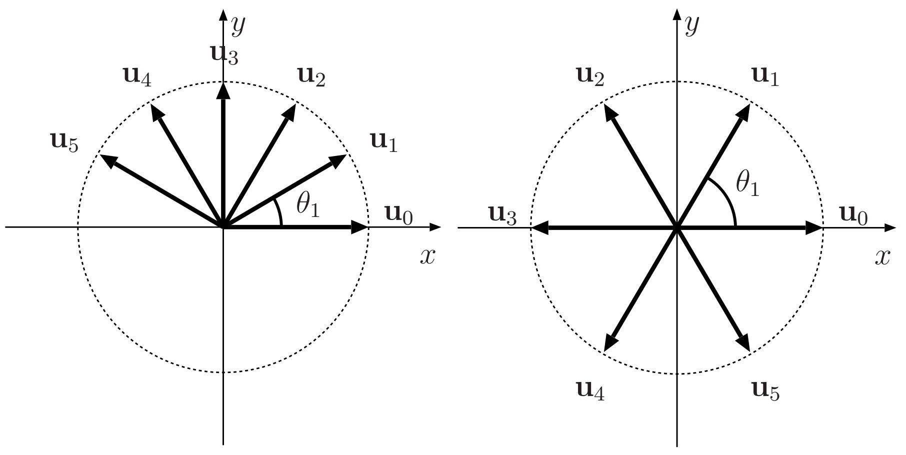
$$M := \sqrt{{g_x}^2 + {g_y}^2}, \tag{2.9}$$

$$\theta := \begin{cases} \cos^{-1}\frac{g_x}{M}, & \text{if } g_y \geq 0 \text{ and } M > 0 \\ 2\pi - \cos^{-1}\frac{g_x}{M}, & \text{if } g_y < 0 \text{ and } M > 0 \\ 0, & \text{if } M = 0 \end{cases}, \tag{2.10}$$

where

$$\mathbf{g} = (g_x,\ g_y) = \left(\frac{\partial I}{\partial x},\ \frac{\partial I}{\partial y}\right). \tag{2.11}$$

Then, the magnitude of each gradient vector is voted to a histogram. The histogram has N orientation bins, where the bins are evenly spread over 0 to π radians as shown in Figure 2.3(a) or 0 to 2π radians as shown in Figure 2.3(b). In Figure 2.3, each $\mathbf{u}_i$ represents an orientation unit vector with magnitude 1. Gradient vector's magnitude M



(a) Half range $[0, \pi)$. (b) Full range $[0, 2\pi)$.

Figure 2.3: Orientation unit vectors $\mathbf{u}_i$. In this figure, six unit vectors are evenly spaced over the half range $[0, \pi)$ (a) or the full range $[0, 2\pi)$ (b).

is voted to the bin including the gradient vector $\mathbf{g}$. Given a histogram of N orientation bins, the $N + 1$ unit vectors, $\mathbf{u}_i (i = 0, \ldots, N)$, are defined as

$$\mathbf{u}_i := \begin{bmatrix} u_i^x \\ u_i^y \end{bmatrix} = \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}, \tag{2.12}$$

where $\theta_i = i\theta_1$ and θ_1 represents the angle between two adjacent unit vectors: θ_1 equals $\frac{\pi}{N}$ for the half range and $\frac{2\pi}{N}$ for the full range.

A basic way of orientation binning is that each gradient vector casts a single vote for a single bin with the range including it or with the nearest orientation as illustrated in Figure 2.4. In the range binning method, unit vectors are used as bin edges as shown in Figure 2.4(a), and the following relation determines the index i^*_{range}:

$$i^*_{\text{range}} \in \{i \mid \theta_i \leq \theta < \theta_{i+1}\}. \tag{2.13}$$

If the half range is adopted and g_y is negative, $\mathbf{g}$ is moved to the first or second quadrant by the point reflection to the origin because unit vectors are located only in $y \geq 0$.

In the nearest binning method, the index of the nearest unit vector, i^*_{nearest}, is calculated as

$$i^*_{\text{nearest}} = \operatorname*{argmax}_i \mathbf{g} \cdot \mathbf{u}_i, \tag{2.14}$$

where unit vectors are located at the bin center as shown in Figure 2.4(b). These two binning methods are equivalent to each other regarding feature description, because their voting index areas are identical, except for the $\frac{\theta_1}{2}$ rotation, as in Figure 2.4. In this dissertation, the sort of HOG using this type of voting is called *NaiveHOG* for convenience.

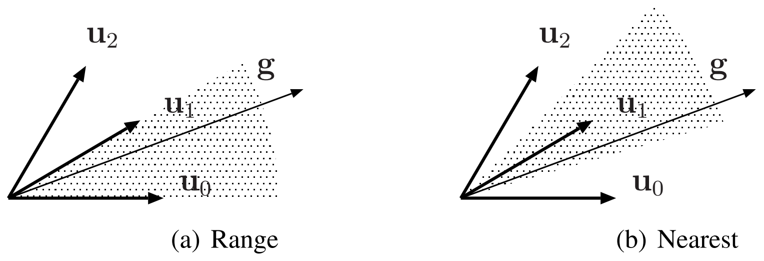
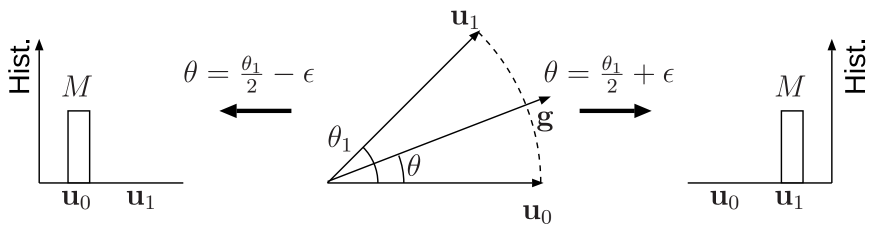


Figure 2.4: Simple orientation binning methods for a gradient vector **g**. In both methods, each gradient vector casts its magnitude for one of the orientations in the shadowed area including **g**.



Hist.
M
$\mathbf{u}_0$ $\mathbf{u}_1$
$\theta = \frac{\theta_1}{2} - \epsilon$
$\mathbf{u}_1$
θ_1
θ
$\mathbf{g}$
$\mathbf{u}_0$
$\theta = \frac{\theta_1}{2} + \epsilon$
M
Hist.
$\mathbf{u}_0$ $\mathbf{u}_1$

Figure 2.5: Aliasing of NaiveHOG, which makes entirely different histograms from similar gradient vectors.

However, as mentioned in [24], there is an aliasing issue, and Figure 2.5 shows an example. Assuming there is a gradient vector in the middle of $\mathbf{u}_0$ and $\mathbf{u}_1$, only a slight difference in θ can make an entirely different voting result. [24] introduces a voting method using bilinear interpolation to resolve this aliasing problem. The concept of using bilinear interpolation is simple. As shown in Figure 2.6, given a gradient vector between two unit vectors, it casts votes for both adjacent bins representing the unit vectors with weights that are proportional to the proximity of the angles, which can be described as the angles. The following equations represent the weights for the adjacent bins, $a_{i^*_{\text{range}}}$, and $b_{i^*_{\text{range}}}$:

$$\begin{gathered} a_{i^*_{\text{range}}} = \frac{\theta_1 - \Delta\theta}{\theta_1} M, \quad b_{i^*_{\text{range}}} = \frac{\Delta\theta}{\theta_1} M, \\ \Delta\theta = \theta - \theta_{i^*_{\text{range}}}. \end{gathered} \tag{2.15}$$

This bilinearly interpolated HOG enables to suppress the influence of aliasing. However, a trigonometric function is necessary for acquiring θ, and bilinear interpolation requires a division and a multiplication, which means complex computations are necessary in hardware implementation. Therefore, in straightforward hardware implementations [5, 6], the HOG calculation is the dominant process from the point of power consumption. Hereafter, this bilinearly interpolated HOG is called *InterHOG*.

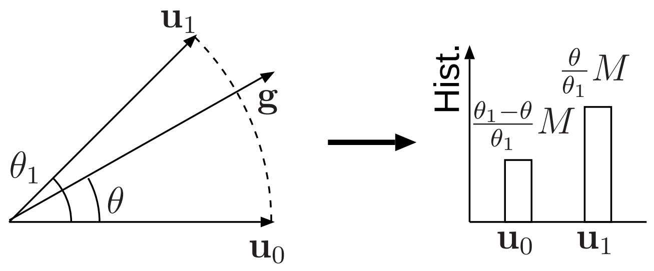


Figure 2.6: Bilinearly interpolated HOG. Given a gradient vector **g**, the interpolated HOG bilinearly decomposes its magnitude M according to θ.

Table 2.1: Coordinates of six unit vectors over the half range ($N = 6$).

	$\mathbf{u}_0$	$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$
u_i^x	1	$\sqrt{3}/2$	$1/2$	0	$-1/2$	$-\sqrt{3}/2$
u_i^y	0	$1/2$	$\sqrt{3}/2$	1	$\sqrt{3}/2$	$1/2$

The magnitude calculation of Eq. (2.9) contains a square root operation, which requires plenty of computational resources in a hardware implementation. Hardware-oriented HOG feature descriptors use approximated magnitude calculations to alleviate this problem. This section explains two approximate magnitude calculation methods used in related works.

Square root approximation (SRA) [54] is a hardware-oriented approximation method for the square root used in [8], and Eq. (2.16) expresses the magnitude calculation:

$$\begin{aligned} M_{\mathrm{SRA}} &:= \max\left((a - 0.125a) + 0.5b, a\right), \\ a &= \max(|g_x|, |g_y|), \quad b = \min(|g_x|, |g_y|). \end{aligned} \tag{2.16}$$

Hardware for the SRA consists of only an adder, a shifter, and a comparator, but the simplification limits its approximation capability. Chen *et al.* proposed a hardware-efficient HOG feature descriptor using SRA [8]. The algorithm calculates SRA and votes $M_{\mathrm{SRA}}/2$ to two adjacent bins to alleviate the aliasing problem. Chen's HOG feature descriptor is called *ChenHOG*.

Another magnitude approximation method is to use the dot product. *HOG-Dot* proposed in [10] adopts the nearest binning method defined in Eq. (2.14) and votes the dot product of **g** and $\mathbf{u}_{i^*_{\mathrm{nearest}}}$ instead of magnitude M. However, in HOG-Dot, the magnitude is voted to the nearest bin and ignores the other weight, which causes information loss.

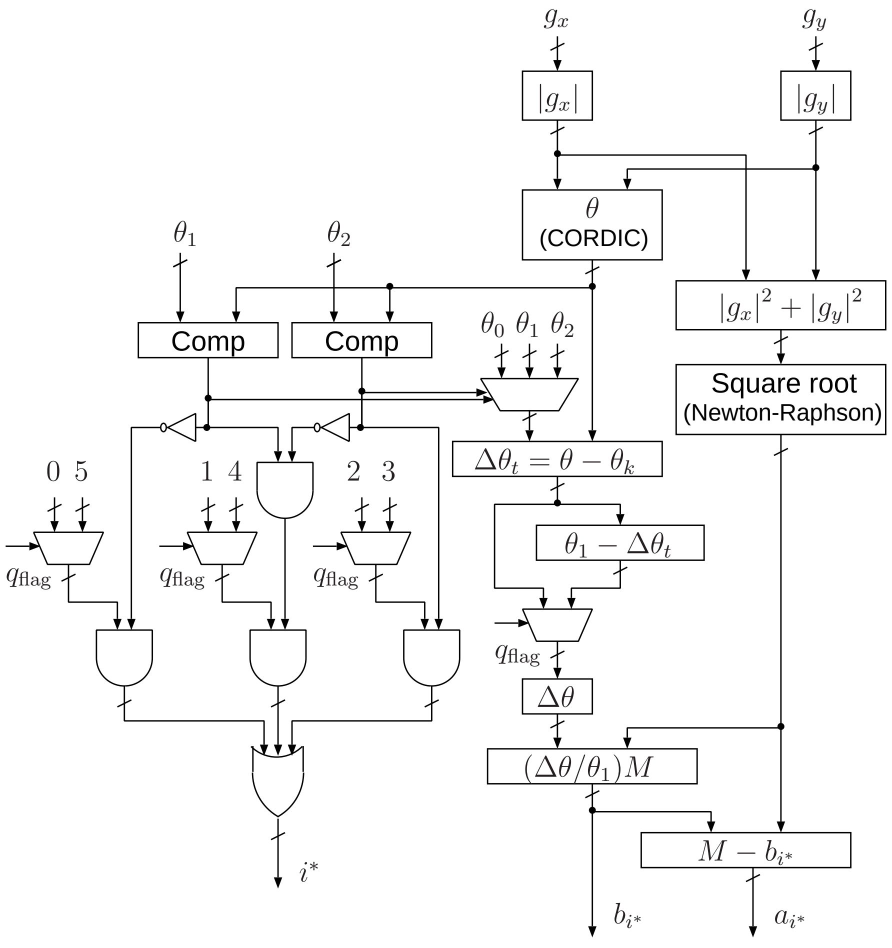


Figure 2.7: Outline of InterHOG hardware.

2.3.2 Hardware Architectures for HOG Feature Descriptors

This section introduces hardware architectures based on the HOG algorithms explained in the previous section. In Chapter 3, they are implemented and used for comparison with the proposed DV-HOG. Their designs are based on their original works but largely improved by the proposed quadrant folding described in Chapter 3. For proving the superiority of DV-HOG, applying quadrant folding to existing HOG algorithms may not be a reasonable choice. However, it makes it possible to focus only on the feature descriptor in the evaluation. Also, even without the advantage of quadrant folding, DV-HOG outperforms existing methods. Each hardware is designed to output a set of index and magnitudes per clock cycle in fixed-point representation and based on the settings of an ACF classifier: six unit vectors over the half range, whose settings are used in the implementation of ACF [36]. In this setting, the unit vectors are located in $y \geq 0$ area, and Table 2.1 summarizes the six unit vectors.

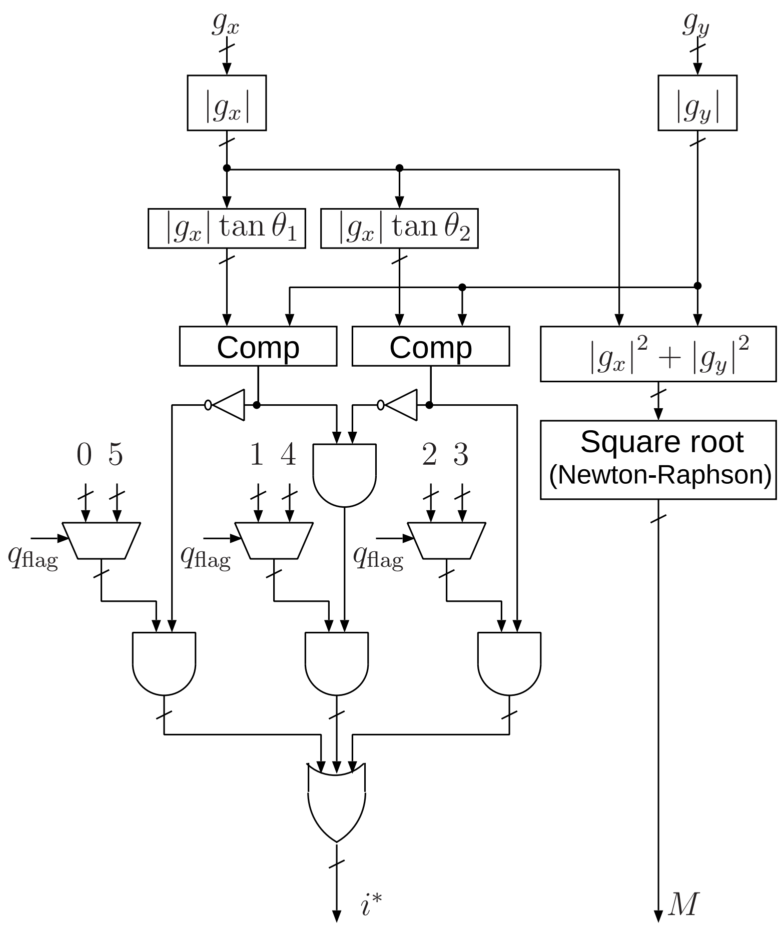


Figure 2.8: Outline of NaiveHOG hardware.

In [5] and [6], Mizuno *et al.* proposed a HOG pedestrian detection hardware using InterHOG. They used CORDIC [55] and the Newton-Raphson method to implement InterHOG. The InterHOG hardware for the evaluation is based on [5] and [6], and the quadrant folding is applied to reduce circuit area. Figure 2.7 shows the InterHOG hardware architecture, where the quadrant flag is used for quadrant folding to represent the signs of the gradient vector, In Figure 2.7, the architecture includes a CORDIC module for the θ calculation, a Newton-Raphson module [56] for the square root calculation, together with multipliers, adders, multiplexers, and simple logic gates. Since the InterHOG hardware adopts straightforward implementation, the processing flow follows the calculation described in the previous section.

The hardware architecture for NaiveHOG is similar to that of InterHOG because NaiveHOG is a simplified InterHOG. NaiveHOG omits internal division, as shown in Figure 2.8.

ChenHOG is the comparison target because it is the smallest hardware

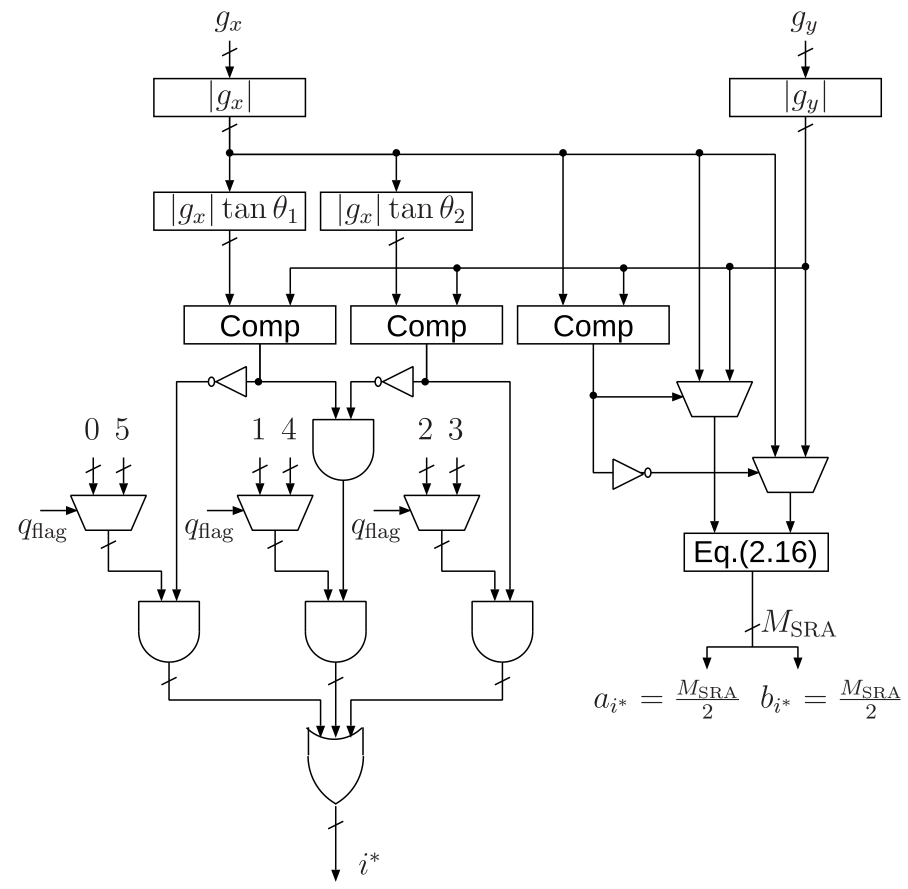


Figure 2.9: Outline of ChenHOG hardware with sign modification.

implementations for HOG calculation, to the best of my knowledge. Figure 2.9 summarizes of the ChenHOG hardware architecture. The hardware mainly consists of two constant multipliers, multiplexers, comparators, and simple logic gates. Note that several multiplexers and comparators used for Eq. (2.16) are omitted in Figure 2.9. The index i^* is selected based on the results of the comparison of $|\mathbf{g}_x| \tan\theta_i$ and $|\mathbf{g}_y|$. Although ChenHOG votes the magnitude to adjacent two bins, no additional divider is required for division by two because the architecture uses fixed-point calculations.

Figure 2.10 shows the hardware architecture for HOG-Dot based on [9]. Because the hardware architecture described in [9] adopts full range, the following preprocessing for mapping to the half range is conducted:

$$g''_x \leftarrow \begin{cases} g_x, & \text{if } g_y \geq 0 \\ -g_x, & \text{otherwise} \end{cases}, \quad g''_y \leftarrow |g_y|. \tag{2.17}$$

The hardware consists of six dot product calculators and ten multiplexers, and each dot product calculator contains two constant multipliers and an adder. After the dot product

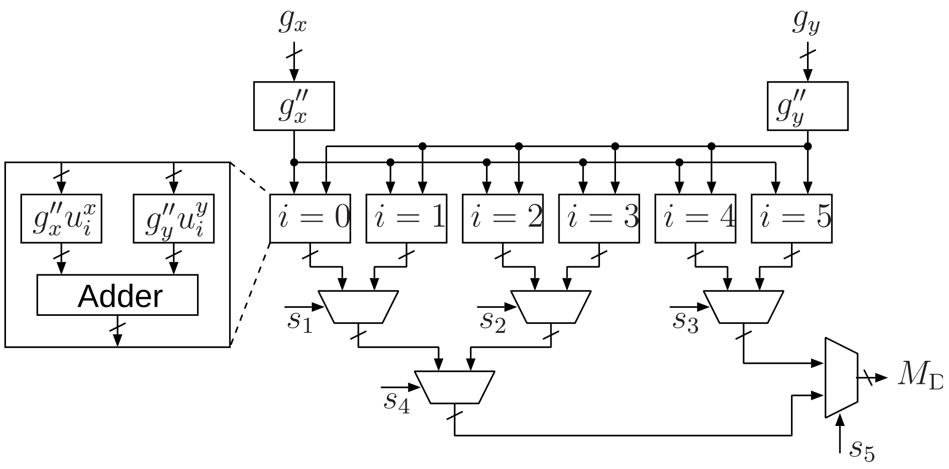


Figure 2.10: Outline of the HOG-Dot hardware proposed in [9].

calculation, each multiplexer selects the larger input value to select the largest one.

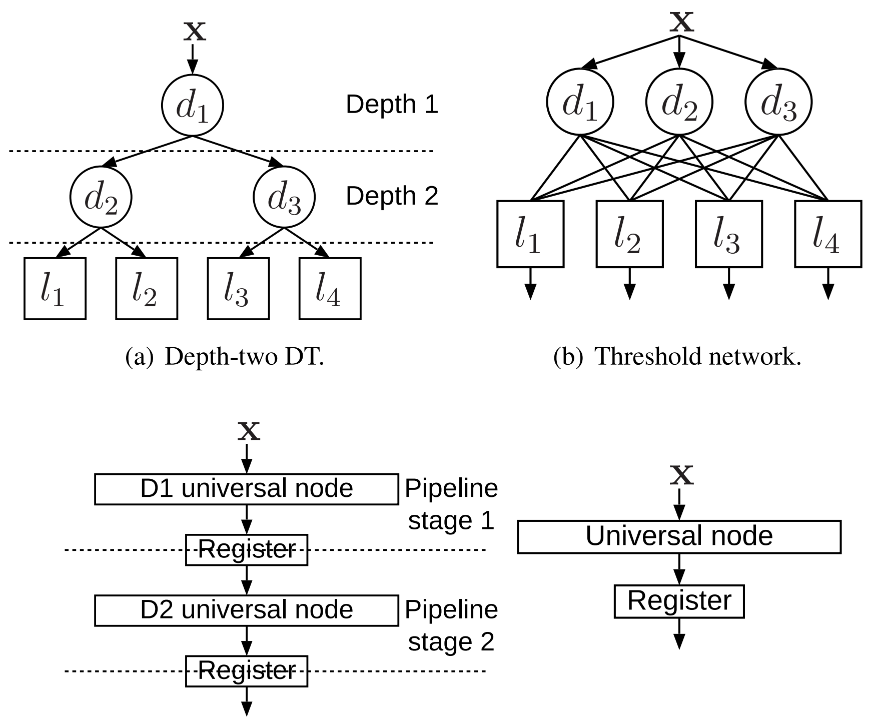
2.4 Boosted Decision Trees and Its Hardware Architectures

BDT is one of ensemble learning methods that use multiple DTs as weak learners in AdaBoost. Each DT consists of decision nodes and leaf nodes, where a decision node selects one of its child nodes based on the comparison result between its input and threshold, and the selected leaf node returns its evaluation value. Given an input feature vector **x**, a BDT classifier is defined as Eq. (2.7). Compared with recent deep learning algorithms, BDT is a shallow machine learning algorithm. However, it is reasonably deep and shows good classification performance for practical applications [57].

2.4.1 Hardware Architectures for Boosted Decision Trees

There exist multiple hardware architectures for BDT [58, 59], and Struharik and Novak classified them into three types in [60]: threshold networks, single-path architectures, and single-node architectures. Figure 2.11(a) is a target depth-two DT, and Figures 2.11(b), 2.11(c), and 2.11(d) show a threshold network, a single-path architecture, and a single-node architecture, which are available implementations for a depth-two DT. The threshold network, shown in Figure 2.11(b), is an architecture that processes all the decision nodes of a DT in parallel, calculating an output O as follows:

$$O = l_1(d_1 d_2) + l_2(d_1 \bar{d}_2) + l_3(\bar{d}_1 d_3) + l_4(\bar{d}_1 \bar{d}_3), \tag{2.18}$$



(a) Depth-two DT. (b) Threshold network.

(c) Single-path architecture. (d) Single-node architecture.

Figure 2.11: BDT and available hardware architectures.

where d_i is the binary response of the i-th decision node, 0 or 1, and l_j is the value of the j-th leaf node. Since threshold networks enable to calculate output instantly after input, it is suitable for applications requiring short time delay between input and output. The single-path architecture, shown in Figure 2.11(c), is an architecture that has pipeline stages of universal nodes, where the number of pipeline stages is equal to the depth of the DT, and the universal nodes are processing elements to carry out the function of decision nodes. Since single-path architectures adopt a pipelined homogeneous structure, it achieves equivalent throughput to the corresponding threshold network with a relatively small amount of hardware resources. The single-node architecture, shown in Figure 2.11(d), also uses universal nodes as processing elements but does not have pipeline stages. The single-node architecture has more flexibility in its design than the others mentioned above in that it can handle any processing order of decision nodes and there exist multiple hardware architectures for storing the responses of decision nodes. However, its processing performance and required hardware resources largely depend on the design. Therefore, for the hardware implementation based on the single-node architecture, the architecture design plays an important role.

Table 2.2: Overview of images in INRIA Person Dataset.

	#Pedestrians	# of positive images	# of negative images
Training	1,208	614	1218
Testing	566	288	453

2.5 Pedestrian Detection

In computer vision, pedestrian detection has been studied for several decades due to (1) difficulty originating from non-rigidity and occlusion and (2) importance in practical situations such as ADAS and surveillance systems. For comparing detection capability between multiple methods, many pedestrian datasets have been used, and INRIA Person Dataset [24] is one of the most well-known datasets [25, 27]. This dissertation mainly uses this dataset for evaluation in Chapters 3, 4, and 5. INRIA Person Dataset contains 2,573 images consisting of 1,832 training images and 741 testing images. Each positive image has annotations indicating locations of pedestrians. Table 2.2 shows the overview of the dataset.

INRIA Person Dataset uses a trade-off between MR and false positive per image (FPPI), and log-average MR as evaluation criteria. The trade-off represents the relationship between false positives and false negatives. Figure 1.2 shows this trade-off as a detection error trade-off (DET) curve of existing object detection methods. If the curve is close to the bottom left, it means the good result. To compare multiple object detection algorithms, log-average MR is used as a representative evaluation criterion. Log-average MR is the log-averaged MRs at nine FPPI points, where the nine points are uniformly sampled in log-space from FPPI equals 0.01 to 1.0. The small log-average MR shows the better result. To compute the DET curve, BDT classifier's threshold is changed, which corresponds to the sign function of Eq. (2.7). Higher threshold generates higher false negatives and lower false positives and vice versa.

2.6 Summary

This chapter explained the preliminary knowledge of visual object detection and its hardware implementation. First, this chapter explained the classifiers for conventional machine learning algorithms. Then, it described the ACF object detection algorithm and its components. Four HOG algorithms are reviewed: the original HOG, a HOG without information loss, and two approximated HOG algorithms for efficient hardware implementation. Next, three types of available hardware architectures of BDT and their characteristics are introduced. Finally, as an evaluation target, pedestrian detection and its dataset are introduced.

Chapter 3

Decomposed Vector Histograms of Oriented Gradients

This chapter explains an information-preserved HOG feature descriptor called DV-HOG for efficient hardware implementation [61]. The basic idea of DV-HOG is vector decomposition of the gradient vector to the adjacent unit vectors. This chapter confirms that DV-HOG is an information-preserved feature descriptor based on the comparison of DV-HOG and InterHOG. Then, it explains a hardware architecture for DV-HOG, and an evaluation result using pedestrian detection shows the improvement of the trade-off between detection accuracy and hardware resources.

3.1 Introduction

The HOG feature descriptor proposed in [24] is still a widely used feature extraction method, and even recently proposed algorithms utilize it or its variants as a feature descriptor [25, 27]. A problem with using HOG feature descriptor in hardware implementation is that it requires large amounts of computational resources for square root and trigonometric functions in Eqs. (2.9) and (2.10). According to [5] and [6], in straightforward hardware implementations using the Newton-Raphson method and CORDIC, HOG feature generation accounts for more than half of the entire power consumption of the dedicated hardware. For that reason, many hardware implementations use simplified HOG-based feature descriptors instead of the original one [7, 8, 11, 12, 48]. These approximations, however, degrade the detection accuracy of the latest object detection algorithms that are sensitive to the representation capability of the feature descriptors.

To solve this problem, this chapter proposes a HOG-based feature descriptor named *decomposed vector HOG*, which utilizes only simple linear algebra for calculation. The most important aspect of this feature descriptor is that its representation capability is equivalent to that of the HOG feature descriptor, but it requires much less computation.

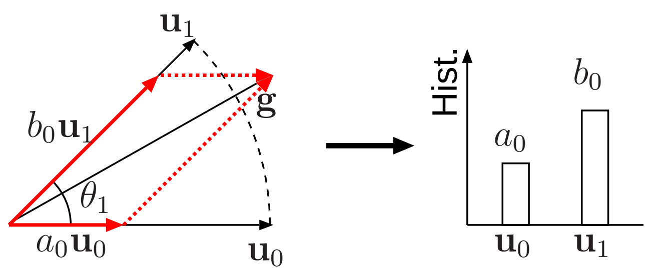


Figure 3.1: Proposed DV-HOG. The proposed method decomposes a gradient vector $\mathbf{g}$ into two adjacent vectors with directions of a pair of $\mathbf{u}_i$, where the vectorial sum of the two decomposed vectors is equal to the gradient vector.

In particular, the method makes it possible to implement existing HOG-based object detection algorithms in a much smaller area and with lower power consumption.

The rest of this chapter is organized as follows. Section 3.2 describes DV-HOG and explains that DV-HOG has a property of information preservation. Section 3.3 describes HOG hardware architecture for DV-HOG. Section 3.4 discusses the results of the evaluation, and Section 3.5 summarizes this chapter.

3.2 Decomposed Vector HOG Feature Descriptor

In contrast to other HOG feature descriptors explained in Section 2.3, the InterHOG feature descriptor is free of any information loss which is the cause of recognition accuracy degradation. However, in the case of InterHOG, ensuring no information loss requires large computational resources. The proposed DV-HOG shares the equivalent representation capability to InterHOG but demands less computation.

3.2.1 Details of DV-HOG

Figure 3.1 shows the concept of DV-HOG. DV-HOG considers a gradient vector $\mathbf{g}$ which is defined in Eq. (2.11) as the vectorial sum of two decomposed vectors:

$$\mathbf{g} = a_i\mathbf{u}_i + b_i\mathbf{u}_{i+1}, \tag{3.1}$$

where $a_i\mathbf{u}_i$ and $b_i\mathbf{u}_{i+1}$ are decomposed vectors with magnitudes a_i and b_i, and they are in the same directions as $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$, respectively. This pair of decomposed vectors composing the gradient vector $\mathbf{g}$ can be calculated as follows. First, in the case of the half range, the following preprocess is conducted to project the third and fourth quadrants to the first and second quadrants if g_y is negative:

$$g_x \leftarrow -g_x, \quad g_y \leftarrow -g_y. \tag{3.2}$$

Algorithm 3.1 DV-HOG (half range).

```
Input: number of bins $N$, gradient vector $\mathbf{g}$
Output: index $i^*$, magnitudes $a_{i^*}, b_{i^*}$
 1: if $g_y < 0$ then
 2:     $g_x \leftarrow -g_x$
 3:     $g_y \leftarrow -g_y$
 4: end if
 5: for $i = 0$ to $\lfloor \frac{N}{2} \rfloor$ do
 6:     $t_i^x = u_i^y g_x$
 7:     $t_i^y = u_i^x g_y$
 8: end for
 9: $b_0 = t_0^y$
10: for $i = 1$ to $N$ do
11:     $b_i = \begin{cases} t_i^y - t_i^x, & \text{if } i \leq \lfloor \frac{N}{2} \rfloor \\ -t_{N-i}^y - t_{N-i}^x, & \text{if } i > \lfloor \frac{N}{2} \rfloor \end{cases}$
12:     $a_{i-1} = -b_i$
13: end for
14: $i^* \in \{i \mid a_i \geq 0 \wedge b_i \geq 0\}$
```

After that, the magnitudes a_i and b_i of the decomposed vectors are calculated as Eq. (3.3), which is derived from Eq. (3.1):

$$a_i = \frac{u_{i+1}^y g_x - u_{i+1}^x g_y}{u_i^x u_{i+1}^y - u_i^y u_{i+1}^x}, \quad b_i = \frac{u_i^x g_y - u_i^y g_x}{u_i^x u_{i+1}^y - u_i^y u_{i+1}^x}, \tag{3.3}$$

where each $\mathbf{u}_i$ is defined in Eq. (2.12), and $\mathbf{u}_N$ is equal to $-\mathbf{u}_0$ for the half range case and $\mathbf{u}_0$ for the full range case. Eq. (3.3) defines all the pairs of decomposed vectors within the range, but a_i and b_i are simultaneously nonnegative if and only if the gradient vector $\mathbf{g}$ is between $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$. When either a_i or b_i is zero, there is more than one index i satisfying the non-negative condition mentioned above, and each i represents identical histogram votes. Finally, the magnitude M_{DV} is used for cell normalization, or M_{DV} is used as a feature in such as [36] and [62]. As novel object detection algorithms use L1 normalization [36, 37, 62], the magnitude in DV-HOG, M_{DV}, is defined as

$$\begin{aligned} M_{\mathrm{DV}} &:= a_{i^*} + b_{i^*}, \\ \text{s.t.} \quad i^* &\in \{i \mid a_i \geq 0 \wedge b_i \geq 0\}, \end{aligned} \tag{3.4}$$

where i^* represents the index of the bin including $\mathbf{g}$.

Algorithm 3.1 is the pseudo code of DV-HOG for the half range case. All u_i^x and u_i^y are constant values, so it is possible to calculate a_i and b_i with constant multiplications. Although there are divisions by $(u_i^x u_{i+1}^y - u_i^y u_{i+1}^x)$ in Eq. (3.3), they are not necessary for practice because the normalization process rescales the feature values. When the half

Table 3.1: Comparison between InterHOG and DV-HOG.

	Magnitude	a_{i^*}	b_{i^*}
InterHOG	M	$\frac{\theta_1-\Delta\theta}{\theta_1}M$	$\frac{\Delta\theta}{\theta_1}M$
DV-HOG	$\frac{\sin\Delta\theta+\sin(\theta_1-\Delta\theta)}{\sin\theta_1}M$	$\frac{\sin(\theta_1-\Delta\theta)}{\sin\theta_1}M$	$\frac{\sin\Delta\theta}{\sin\theta_1}M$

range is adopted, unit vectors in the first and second quadrants are linear symmetry to the y-axis. By using this symmetry, the number of the loops in lines 5–8 of Algorithm 3.1 can be reduced to half: focusing on unit vectors in the first quadrant is enough for calculation. Considering there exist $\lfloor \frac{N}{2} \rfloor$ unit vectors in the first quadrant, the number of multiplications for each is $2 \times \lfloor \frac{N}{2} \rfloor$. Moreover, due to the similarity between the calculations of a_i and b_i, a_i can be obtained by reusing b_i and vice versa: for the full range,

$$a_i = -b_j \quad (j = i + 1 \bmod N), \tag{3.5}$$

and for the half range,

$$a_i = \begin{cases} -b_{i+1}, & i = 0, \ldots, N-2 \\ b_0, & i = N-1 \end{cases}. \tag{3.6}$$

As a result, the number of computations can be reduced to a quarter by exploiting the symmetry of the unit vectors and the similarity of a_i and b_i.

3.2.2 Representation Capability Analysis

The representation capability of DV-HOG is equivalent to that of InterHOG in that neither causes any information loss unlike other HOG algorithms described in Section 2.3: $\mathbf{g}$ can be reproduced by using a_i and b_i defined in Eqs. (2.15) and (3.3) for InterHOG and DV-HOG, respectively. However, the representations of InterHOG and DV-HOG are slightly different. Therefore, this section compares DV-HOG and InterHOG and provides an analysis of their representation capabilities.

To compare DV-HOG with InterHOG directly, Eq. (3.7) is used to rewrite the gradient vector's components:

$$g_x = M\cos\theta, \quad g_y = M\sin\theta. \tag{3.7}$$

Let $a_{i^*}^{\mathrm{DV}}, b_{i^*}^{\mathrm{DV}}$ and $a_{i^*}^{\mathrm{Inter}}, b_{i^*}^{\mathrm{Inter}}$ be the voted weights of DV-HOG and InterHOG. Then, $a_{i^*}^{\mathrm{DV}}$ and $b_{i^*}^{\mathrm{DV}}$ are rewritten by substituting Eq. (2.11) for Eq. (3.3):

$$a_{i^*}^{\mathrm{DV}} = \frac{\sin(\theta_1 - \Delta\theta)}{\sin\theta_1}M, \quad b_{i^*}^{\mathrm{DV}} = \frac{\sin\Delta\theta}{\sin\theta_1}M, \tag{3.8}$$

where

$$\sin\theta_1 = u_i^x u_{i+1}^y - u_i^y u_{i+1}^x \tag{3.9}$$

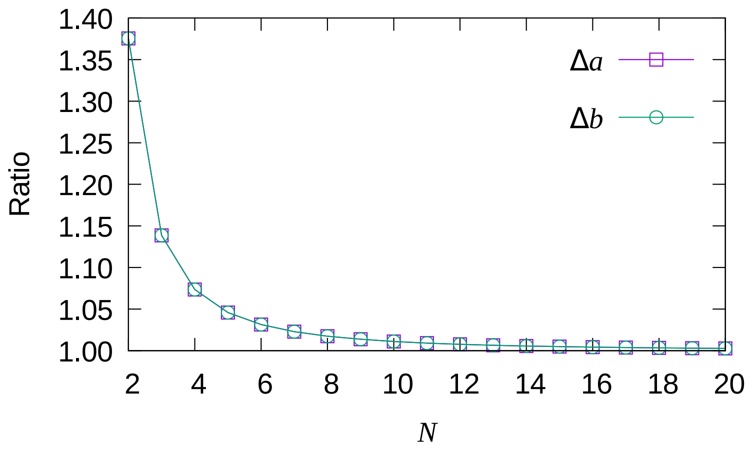


Figure 3.2: Comparison of voted weights of DV-HOG and InterHOG.

and $\Delta\theta$ is the same variable defined in Eq. (2.15). Table 3.1 compares DV-HOG and InterHOG by Eqs. (2.15) and (3.8). Here, it is easily seen that DV-HOG and InterHOG have different magnitudes for the decomposed vectors, but share a similar tendency toward $\Delta\theta$. Let Δa and Δb be the differences between DV-HOG and InterHOG of a_{i^*} and b_{i^*}:

$$\Delta a := a_{i^*}^{\mathrm{DV}} - a_{i^*}^{\mathrm{Inter}} \quad \text{and} \quad \Delta b := b_{i^*}^{\mathrm{DV}} - b_{i^*}^{\mathrm{Inter}}. \tag{3.10}$$

The following equation holds in the limit as θ_1 approaches 0:

$$\lim_{\theta_1 \to 0} \Delta b = \lim_{\theta_1 \to 0} \frac{\sin \Delta\theta}{\sin \theta_1} M - \lim_{\theta_1 \to 0} \frac{\Delta\theta}{\theta_1} M = 0, \tag{3.11}$$

because the second term of Eq. (3.11) is equal to

$$\lim_{\theta_1 \to 0} \frac{r\theta_1}{\theta_1} M = rM \quad \text{s.t.} \quad r \in [0, 1), \tag{3.12}$$

and the first term is equal to

$$\lim_{\theta_1 \to 0} \frac{\sin r\theta_1}{\sin \theta_1} M = \lim_{\theta_1 \to 0} \frac{\sin r\theta_1}{r\theta_1} \frac{r\theta_1}{\sin \theta_1} M = rM, \tag{3.13}$$

where $r\theta_1 = \Delta\theta$. In the same way, the following Eq. (3.14) can be proven as well:

$$\lim_{\theta_1 \to 0} \Delta a = \lim_{\theta_1 \to 0} \frac{\sin(\theta_1 - \Delta\theta)}{\sin \theta_1} M - \lim_{\theta_1 \to 0} \frac{\theta_1 - \Delta\theta}{\theta_1} M = 0. \tag{3.14}$$

Let us compare how different the voted weights between DV-HOG and InterHOG over N, where the difference is calculated at the point $\theta^{(a)}$ for a_i and $\theta^{(b)}$ for b_i, which

Table 3.2: Comparison of HOG algorithms.

Algorithm	i^*	a_{i^*}	b_{i^*}
DV-HOG	$\left\lfloor \frac{\theta}{\theta_1} \right\rfloor$	$\frac{\sin(\theta_1 - \Delta\theta)}{\sin \theta_1} M$	$\frac{\sin \Delta\theta}{\sin \theta_1} M$
InterHOG [24]	$\left\lfloor \frac{\theta}{\theta_1} \right\rfloor$	$\frac{\theta_1 - \Delta\theta}{\theta_1} M$	$\frac{\Delta\theta}{\theta_1} M$
NaiveHOG [24]	$\left\lfloor \frac{\theta}{\theta_1} \right\rfloor$	M	0
ChenHOG [8]	$\left\lfloor \frac{\theta}{\theta_1} \right\rfloor$	$\frac{M_{\mathrm{SRA}}}{2}$	$\frac{M_{\mathrm{SRA}}}{2}$
HOG-Dot [10]	$\operatorname*{argmax}_i \ \mathbf{g} \cdot \mathbf{u}_i$	$\mathbf{g} \cdot \mathbf{u}_{i^*}$	0

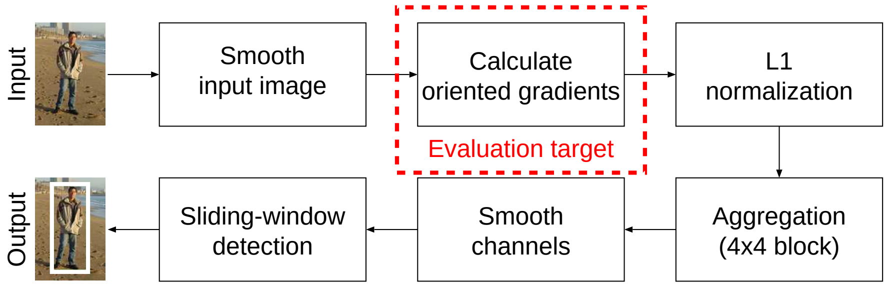


Figure 3.3: Evaluation target process in ACF [36].

gives the maximum difference:

$$\theta^{(a)} := \operatorname*{argmax}_{0 \le \theta < \theta_1} \Delta a = \theta_1 - \cos^{-1} \frac{\sin \theta_1}{\theta_1}, \tag{3.15}$$

$$\theta^{(b)} := \operatorname*{argmax}_{0 \le \theta < \theta_1} \Delta b = \cos^{-1} \frac{\sin \theta_1}{\theta_1}. \tag{3.16}$$

Figure 3.2 shows the behavior expected from Eqs. (3.11) and (3.14): as N increases, the voted weights of DV-HOG and InterHOG converge. In particular, for a practical N, such as 6 or 9, the difference in voted weights is a few percents at most.

3.2.3 Detection Accuracy Evaluation

To evaluate the influence of the DV-HOG feature descriptor on object detection accuracy, DV-HOG is used in an ACF classifier [36]. Figure 3.3 summarizes the processing flow of the ACF classifier, and here, this section focuses on the oriented gradients calculation. Although the C++ software implementation is based on Dollár's MATLAB code available online [1], the calculations of oriented gradients and magnitude

[1]https://github.com/pdollar/toolbox

Table 3.3: Pedestrian detection parameters for the ACF classifier.

Parameter	Symbol	Value
Pixel Step	S_{pixel}	4
Scale Step	S_{scale}	$2^{1/8}$
Window Width	W_{win}	48
Window Height	H_{win}	96

Table 3.4: Log-average MR comparison on INRIA Person Dataset.

Method	Log-avg. MR
ACF-Exact [36] + DV-HOG	**16.24%**
ACF-Exact [36] + HOG-Dot [10]	17.57%
ACF-Exact [36] + InterHOG [24]	17.90%
ACF-Exact [36] + NaiveHOG [24]	20.60%
ACF-Exact [36] + ChenHOG [8]	21.42%

values are varied for each HOG feature descriptor. Table 3.2 summarizes the feature descriptors used in the evaluation. The evaluation uses 32-bit floating point representation for g_x, g_y, a_{i^*}, and b_{i^*}, and 3-bit fixed point representation for i^*. Fast feature pyramids (FFP) [36] are not used to exclude its impact on detection accuracy. Detection accuracy is evaluated on the public pedestrian detection datasets, INRIA Person Dataset [24], and KITTI Vision Benchmark Suite [63].

First, INRIA Person Dataset [24] is used for the training and testing procedures, and the well-known pedestrian detection benchmark [25] is used for evaluation. Table 3.3 lists the object detection parameters used in the evaluation. Table 3.4 shows the log-average MR, and Figure 3.4 shows the DET curves of each HOG algorithm. We can see that DV-HOG gives equivalent or better DET compared with the other HOG feature descriptors. Moreover, Table 3.4 shows that its log-average MR, 16.24%, is around 5% better than the approximated method, ChenHOG. Although Chen *et al.* indicated that the detection accuracy of ChenHOG is equivalent to that of NaiveHOG based on a false positive per window evaluation in [8], ChenHOG has approximately 0.8% worse log-average MR compared with NaiveHOG in the FPPI-based evaluation. HOG-Dot slightly degrades detection accuracy compared with InterHOG and DV-HOG which have no information loss.

In the evaluation using the KITTI Vision Benchmark Suite, the 7,481 training data are split to 6,000 for training and 1,481 for testing because the ground truth for test data are not publicly available. Table 3.3 lists the parameters used in the evaluation, and Table 3.5 shows the average precision (AP) on pedestrian detection. AP is the average of precision values corresponding to recall at $0.0, 0.1, \dots, 1.0$. Training data are different; nevertheless, the AP reported in Table 3.5 shows the equivalent accuracy to that reported

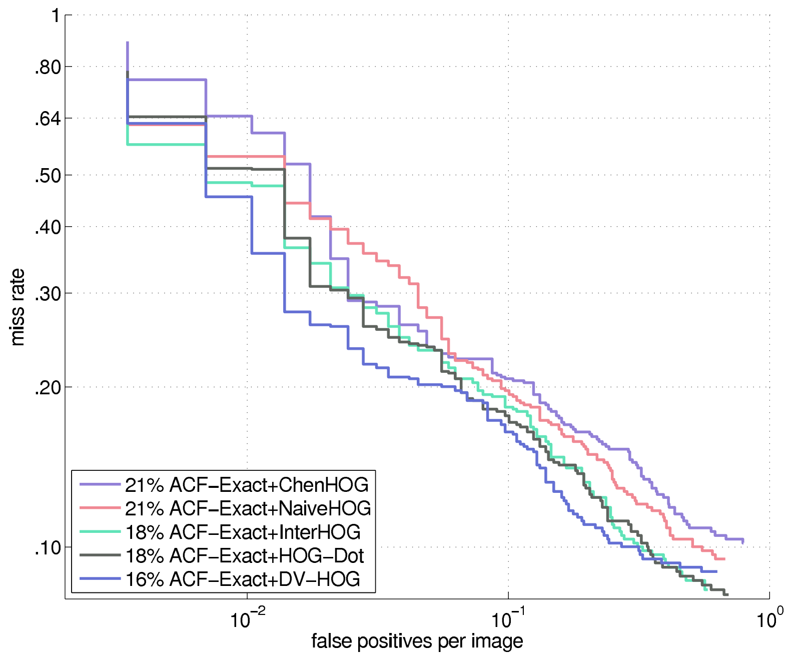


Figure 3.4: DET curves of five ACF classifiers using different HOG feature descriptors. The closer a curve gets to the bottom left corner, the better the DET is.

in KITTI Evaluation [2]. The result indicates the similar tendency to the INRIA Person Dataset. DV-HOG and InterHOG, which are HOG algorithms without approximation, show higher AP than others.

Information loss in HOG calculation has a considerable influence on detection accuracy especially in a noisy environment such as practical applications. For evaluation, input image smoothing is disabled to evaluate the detection accuracy on INRIA Person Dataset. As shown in Table 3.6, log-average MR increases from 9.3% to 11.9% without denoising. The result indicates that DV-HOG achieves the best detection

[2]http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d

Table 3.5: Average precision comparison on pedestrian detection using KITTI Vision Benchmark Suite.

Method	Easy	Moderate	Hard
ACF (reported in KITTI Evaluation)	49.08%	40.62%	36.66%
ACF-Exact [36] + DV-HOG	**48.46%**	**40.07%**	**34.02%**
ACF-Exact [36] + InterHOG [24]	48.14%	39.72%	33.31%
ACF-Exact [36] + HOG-Dot [10]	47.83%	38.80%	32.82%
ACF-Exact [36] + ChenHOG [8]	44.77%	36.74%	31.13%
ACF-Exact [36] + NaiveHOG [24]	41.50%	34.78%	29.36%

Table 3.6: Log-average MR comparison without smoothing image on INRIA Person Dataset.

Method	Log-avg. MR	Diff. from Table 3.4
ACF-Exact [36] + DV-HOG	**25.67%**	+9.44%
ACF-Exact [36] + InterHOG [24]	27.25%	+9.35%
ACF-Exact [36] + HOG-Dot [10]	28.31%	+10.74%
ACF-Exact [36] + NaiveHOG [24]	29.90%	+9.30%
ACF-Exact [36] + ChenHOG [8]	33.35%	+11.93%

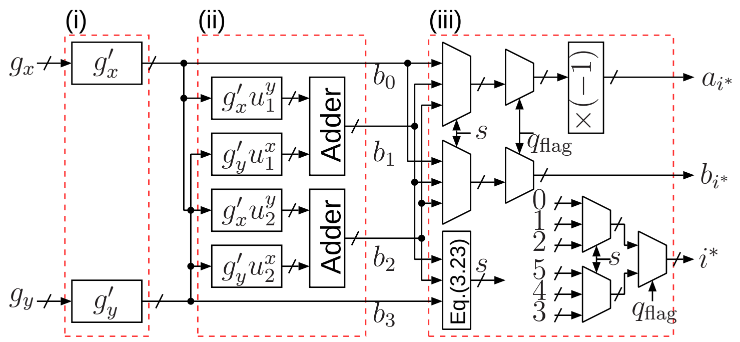
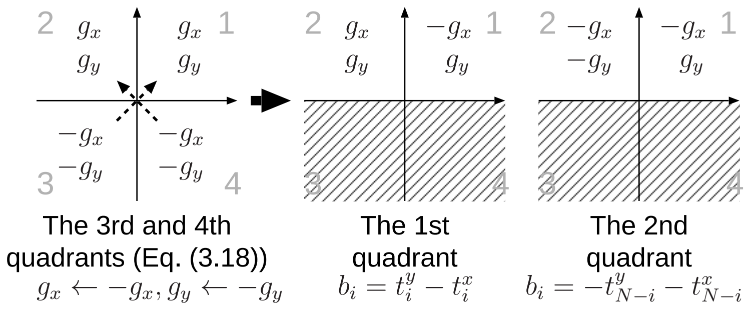


Figure 3.5: Outline of DV-HOG hardware.

accuracy without denoising, and its accuracy degradation is small compared with other methods. The differences in log-average MR using DV-HOG and HOG-Dot are 9.4% and 10.7%, respectively.

3.3 Hardware Architectures for HOG Feature Descriptors

So far, this chapter has explained that DV-HOG has the equivalent representation capability to InterHOG, and because it does not have any complex computations, it would be a useful implementation in hardware. This section proposes a DV-HOG hardware architecture used for the evaluation. Similar to the hardware architectures explained in Section 2.3.2, the hardware is designed to output a set of index and magnitudes per clock cycle in fixed-point representation and based on the settings of an ACF classifier: six unit vectors over the half range. This section also introduces *quadrant folding* to reduce hardware resource utilization, which can be applied to any HOG feature descriptors.



2
1
3
4
g_x
g_y
$-g_x$
$-g_y$
The 3rd and 4th quadrants (Eq. (3.18))
$g_x \leftarrow -g_x, g_y \leftarrow -g_y$
The 1st quadrant
$b_i = t_i^y - t_i^x$
The 2nd quadrant
$b_i = -t_{N-i}^y - t_{N-i}^x$

Figure 3.6: Sign inversion of g_x and g_y for quadrant folding.

3.3.1 Hardware Architecture for DV-HOG

Figure 3.5 summarizes the DV-HOG hardware architecture. The hardware consists of three parts: (i) quadrant folder for mapping **g** to the first quadrant, (ii) vector decomposer for each unit vector in the first quadrant, and (iii) selector for the output. Because a_i can be calculated from b_i as explained in Algorithm 3.1, this hardware calculates b_i only.

The quadrant folder aims to calculate b_i by the single equation regardless of the position of **g**. Eq. (3.17) shows its calculation consisting of two multiplication with constant and one addition using **u** in the first quadrant:

$$b_i = g'_y u_i^x + g'_x u_i^y, \tag{3.17}$$

where $\mathbf{g}' = (g'_x,\ g'_y)$ is used to modify the sign of **g** as shown in Figure 3.6. The following explains the steps of $\mathbf{g}'$ calculation. First, if **g** is in the third or fourth quadrant, **g** is projected to the first or second quadrant by the point symmetry to the origin. This is because b_i does not change due to the point symmetry of the unit vectors. The movement is formulated as follows:

$$g_x \leftarrow -g_x, \quad g_y \leftarrow -g_y, \tag{3.18}$$

which corresponds to lines 1 to 4 in Algorithm 3.1. Then, if **g** is in the second quadrant, **g** is mapped to the first quadrant based on the linear symmetry of **u** to the y-axis. The conversion is represented as follows:

$$\begin{aligned} g'_y &\leftarrow -g_y. \qquad (3.19) \\ \because b_i &= g'_y(-u_i^x) + g'_x u_i^y = (-g'_y)u_i^x + g'_x u_i^y \end{aligned}$$

Now, **g** is in the first quadrant, and by comparing the signs of **g** and $\mathbf{g}'$ in Eqs. (3.3) and (3.17), it is necessary to modify the sign of g_x as follows:

$$g'_x \leftarrow -g_x. \tag{3.20}$$

Table 3.7: Select signals for each bin.

i^*	0	1	2	3	4	5
s_1, s_0	01	10	00	00	10	01
q_{flag}	0	0	0	1	1	1

Therefore, the following sign conversions summarize the quadrant folding:

$$\begin{aligned} g'_x &\leftarrow \begin{cases} |g_x|, & \text{if } q_{\text{flag}} = 1 \\ -|g_x|, & \text{otherwise} \end{cases}, \\ g'_y &\leftarrow \begin{cases} |g_y|, & \text{if } q_{\text{flag}} = 0 \\ -|g_y|, & \text{otherwise} \end{cases}, \end{aligned} \tag{3.21}$$

where

$$q_{\text{flag}} := \text{sign}(g_x) \oplus \text{sign}(g_y), \tag{3.22}$$

$\oplus$ is an exclusive OR operator, and the sign function is defined in Eq. (2.5).

In the vector decomposition part, it is possible to calculate each b_i with Eq. (3.17). Eq. (3.17) is implemented with adders in Figure 3.5, in which there is no adder for $\mathbf{u}_0$ or $\mathbf{u}_3$ because they are unit vectors on the x and y-axes. Moreover, in the case of ACF, the multiplication of u_1^y and u_2^x, which is $1/2$, does not require a multiplier, and the implementation makes use of this optimization.

The last part of the DV-HOG hardware selects the index i^* and its weights a_{i^*} and b_{i^*}, which corresponds to line 14 in Algorithm 3.1. For each of the three outputs, three multiplexers are used as shown in Figure 3.5: two multiplexers in the first layer and a multiplexer in the second layer. The first layer selects a bin in each quadrant, and the second layer selects the quadrant based on the signs of $\mathbf{g}$. The select signal s for multiplexers in the first layer is a 2-bit signal, and each bit is defined as follows:

$$\begin{aligned} s_0 &:= \text{sign}(b_0) \oplus \text{sign}(b_1), \\ s_1 &:= \text{sign}(b_1) \oplus \text{sign}(b_2). \end{aligned} \tag{3.23}$$

The select signal for the second layer is q_{flag} defined in Eq. (3.22). Table 3.7 shows the select signals for each bin. Table 3.8 summarizes the a_{i^*} and b_{i^*} values for each corresponding bin, which is calculated by using Eq. (3.6) and the following equation

$$b_i = b_{6-i}. \tag{3.24}$$

3.3.2 Application of Quadrant Folding to HOG-Dot

This section introduces *quadrant folding* to reduce hardware resource utilization, which uses the symmetry about the x and y-axes. Quadrant folding utilizes $\mathbf{u}_i$ or θ_i of the

Table 3.8: Coefficients of DV-HOG after sign modification.

i^*	0	1	2	3	4	5
a_{i^*}	$-b_1$	$-b_2$	$-b_3$	$-b_2$	$-b_1$	$-b_0$
b_{i^*}	b_0	b_1	b_2	b_3	b_2	b_1

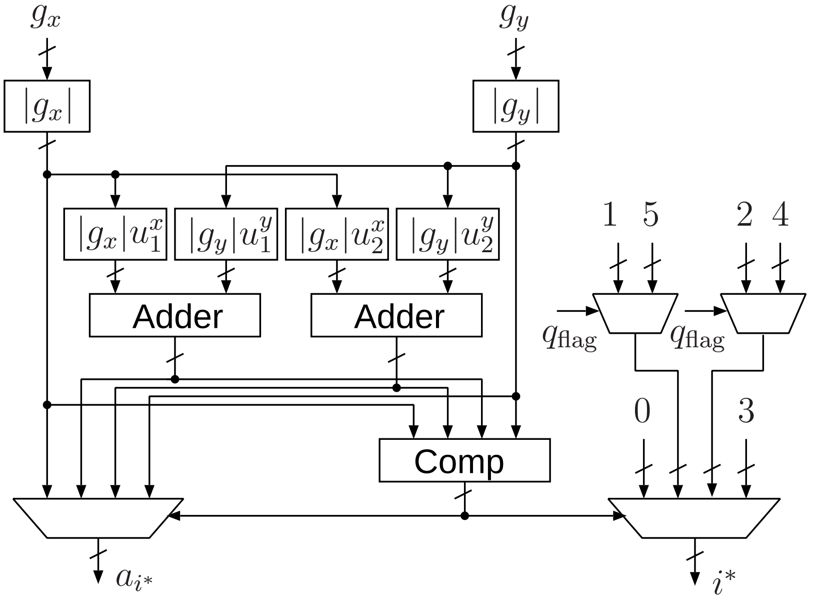


Figure 3.7: Outline of HOG-Dot hardware with quadrant folding.

first quadrant to minimize the circuit scale, and each architecture outputs its calculation results based on the signs of g_x and g_y.

The hardware architecture proposed in [9] calculates for all the quadrants. However, the quadrant folding, a part of the proposed HOG algorithm, can be also applied to HOG-Dot to reduce the hardware resources. Figure 3.7 outlines the hardware architecture of HOG-Dot with the quadrant folding, which is called *modified HOG-Dot*. Compared with Figure 2.10, the comparison candidate is reduced from six to four, and the number of the hardware resource is two adders, a comparator, and five multiplexers. The hardware architecture for HOG-Dot with quadrant folding is similar to that of DV-HOG in that dot product calculations for $\mathbf{u}_0$ and $\mathbf{u}_3$ are omitted.

3.4 Evaluation

The evaluation compares the hardware implementations for DV-HOG, InterHOG, NaiveHOG, ChenHOG, HOG-Dot, and modified HOG-Dot, where HOG-Dot is a comparison baseline to distinguish the effect of the proposed quadrant folding. Each

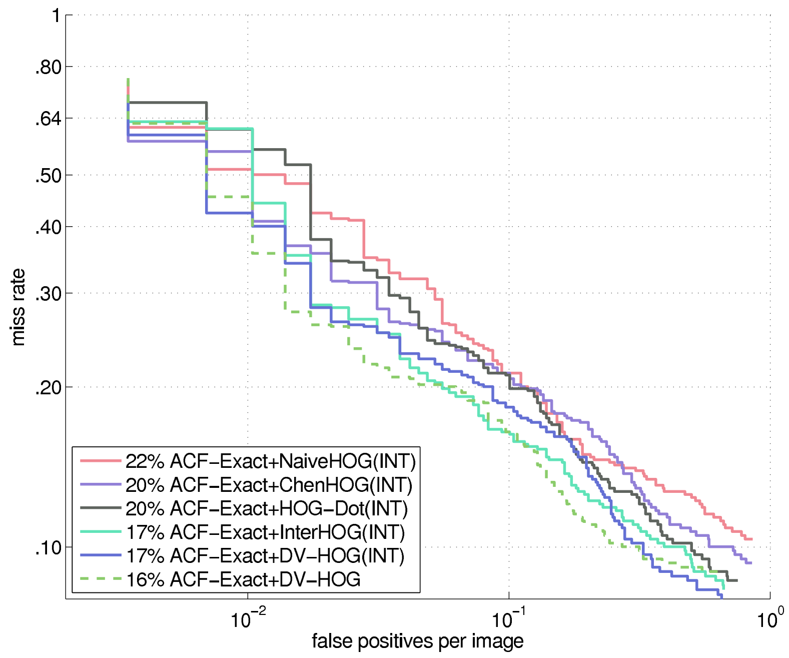


Figure 3.8: DET curves of fixed-point ACF classifiers.

HOG algorithm is evaluated in fixed-point arithmetic from the aspect of the algorithm and the hardware. First, fixed-point ACF software implementation is used for detection accuracy evaluation. Then, for logic synthesis and FPGA implementation, each HOG module is implemented using Verilog hardware description language (HDL).

Because the ACF classifier [36] uses floating LUV values in the range of $[0, 1]$ as the input of the HOG calculation, floating LUV values are converted into integers in the range of $[0, 255]$ as follows:

$$I_{\text{int}} = \lfloor I_{\text{float}} \times 255 + 0.5 \rfloor, \tag{3.25}$$

where I_{float} and I_{int} are floating and integer values, respectively. Prewitt filtering is used in the detection accuracy evaluation, but it is not included in hardware implementations since all HOG methods use it. As a result, each hardware implementation uses two 9-bit fixed-point signed input signals for g_x and g_y. The following equation is used for the fixed-point number representation of the parameters:

$$m_k(x) = \lfloor x \times 2^k + 0.5 \rfloor, \tag{3.26}$$

where k is the bit width of the fractional part.

3.4.1 Detection Accuracy Evaluation with Fixed-point Arithmetic

A software implementation is used for evaluating the detection accuracy with fixed-point arithmetic. The evaluation uses the 4-bit fixed-point arithmetic. To compare the

Table 3.9: Log-average MR comparison with fixed-point representation.

Method	Log-avg. MR	Diff. from Table 3.4
ACF-Exact [36] + DV-HOG (INT)	**17.06%**	+0.82%
ACF-Exact [36] + InterHOG [24] (INT)	17.49%	-0.41%
ACF-Exact [36] + HOG-Dot [10] (INT)	20.24%	+2.67%
ACF-Exact [36] + ChenHOG [8] (INT)	20.45%	-0.97%
ACF-Exact [36] + NaiveHOG [24] (INT)	21.88%	+1.28%

Table 3.10: Logic synthesis settings.

Logic synthesis tool	Design Compiler Version I-2013.12
Cell library	NanGate 45nm Open Cell Library (typical)
Target frequency	200 MHz

result with the floating-point arithmetic shown in Table 3.4, the floating-point classifiers are converted to the fixed-point classifiers based on the quantization method which will be proposed in Chapter 4.

Figure 3.8 shows the DET curves for each fixed-point implementation and DV-HOG's floating-point implementation. Fixed-point DV-HOG shows better log-average MR compared with other fixed-point HOG implementations. Also, the difference of detection accuracy of floating-point DV-HOG and that of fixed-point is less than 1%: DV-HOG is robust to the conversion from floating-point to fixed-point. Table 3.9 shows the detailed log-average MRs for each fixed-point implementation and the log-average MR difference from Table 3.4. HOG calculations without information loss, which are DV-HOG and InterHOG, achieve more than 2.7% lower log-average MR than other methods.

Table 3.11: Cell-based synthesis comparison of DV-HOG and conventional methods.

Method	Cell Area (μm^2)	# of Nand Gates
ChenHOG [8]	479.60	601
Modified HOG-Dot	572.43	717
DV-HOG	685.48	859
HOG-Dot [9]	1,194.07	1,496
NaiveHOG [24]	1,606.64	2,014
InterHOG [24]	9,724.43	12,186

NAND2_X1: 0.798 μm^2

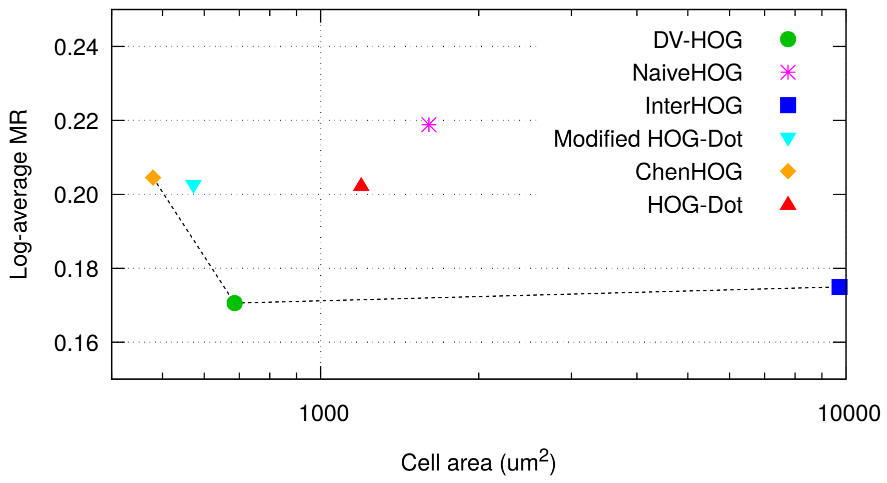


Figure 3.9: Log-average MR and the circuit area of HOG algorithms.

3.4.2 Area Evaluation with Logic Synthesis

Verilog HDL implementation of each HOG module was synthesized to the gate level by using a logic synthesis tool to compare the circuit scales. Table 3.10 describes the synthesis settings. The results shown in Table 3.11 indicate that ChenHOG had the smallest cell area, as expected. In the InterHOG hardware, the large cell area results from its complex modules, such as CORDIC and Newton-Raphson, and the registers for reducing critical path and synchronizing output signals. The comparison between two HOG-Dot hardware implementations shows that the quadrant folding reduces the cell area to around half. Although HOG-Dot with quadrant folding has a similar hardware structure to that of DV-HOG, HOG-Dot with quadrant folding is slightly smaller than DV-HOG because the calculation in HOG-Dot uses unsigned numbers while DV-HOG uses signed numbers.

Figure 3.9 plots the correlation between cell area reported in Table 3.11 and log-average MR shown in Table 3.9. In the figure, the closer the point gets to the bottom left corner, the better the result is. Figure 3.9 indicates that DV-HOG achieves equivalent log-average MR to InterHOG with 14.2 times smaller cell area. Although the modified HOG-Dot with quadrant folding and ChenHOG are better in terms of the cell area, DV-HOG shows better detection accuracy with a slight increase in cell area. This improved trade-off could be an advantage for practical applications.

3.4.3 FPGA Implementation

The HOG algorithms described in Sections 2.3.2 and 3.3 were also implemented in an FPGA. Although some of the algorithms reported in the literature provide FPGA

Table 3.12: FPGA implementation settings.

Target device	Xilinx xc7k325t-2ffg900
Synthesis tool	Vivado 2015.4.2
Target frequency	100MHz

Table 3.13: FPGA implementation results.

Method	Slice LUTs		Slice registers	
ChenHOG [8]	128	(0.06%)	15	(<0.01%)
Modified HOG-Dot	152	(0.07%)	19	(<0.01%)
DV-HOG	194	(0.10%)	33	(<0.01%)
NaiveHOG [24]	269	(0.13%)	95	(0.02%)
HOG-Dot [9]	356	(0.17%)	19	(<0.01%)
InterHOG [24]	1,060	(0.52%)	447	(0.11%)

implementations, their evaluations were carried out with different settings, making a comparison difficult. Thus, this section compares each algorithm in an FPGA with the settings described in Table 3.12. Table 3.13 shows the results. The result shows that DV-HOG requires only 18% of LUTs and 7% of slice registers compared with InterHOG. The number of LUTs used in the FPGA implementation shows a similar tendency in the cell area as the logic synthesis evaluation.

3.4.4 Discussion

For many object detection hardware implementations, high detection accuracy is the primary requirement. However, the cell area is also important because it influences on the power consumption of the system. Although this chapter discussed a single HOG module, it is necessary to use multiple HOG modules in parallel for practical applications. Figure 3.10 shows a hardware architecture for multi-scale object detection based on [64]. This hardware adopts a hybrid pyramid generation approach for image and classifier scaling: an input image is scaled to octave images and multiple classifiers are applied to each image, where each classifier finds the different size of target objects. This approach enables to reduce the number of feature extraction for fast detection and the number of classifiers for small memory requirements. The hardware consists of scalers, feature extractors, classifiers of boosted decision trees, and an NMS module. Each scaler resizes an image to a quarter, and each classifier module applies multiple sizes of classifiers to features in order, where each classifier uses a sliding-window approach. Finally, NMS converges detection results from each classifier.

To evaluate the influence of HOG hardware implementation to the object detection system, the number of required HOG modules, N_{HOG} is calculated. Consider an object detection system for $W_{\mathrm{img}} \times H_{\mathrm{img}}$ images in N_{fps} fps video streams at an operation

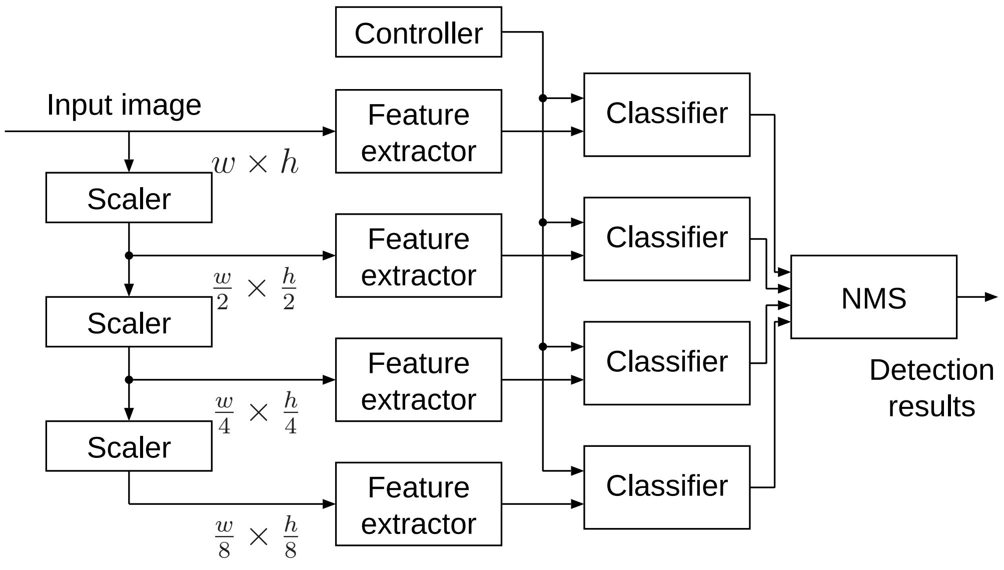

Controller
Input image
$w \times h$
Scaler
Feature extractor
Classifier
$\frac{w}{2} \times \frac{h}{2}$
$\frac{w}{4} \times \frac{h}{4}$
$\frac{w}{8} \times \frac{h}{8}$
NMS
Detection results

Figure 3.10: Outline of hardware architecture for object detection.

frequency f. In object detection using a sliding-window method, the number of pixels computed in HOG modules for a $w \times h$ image is

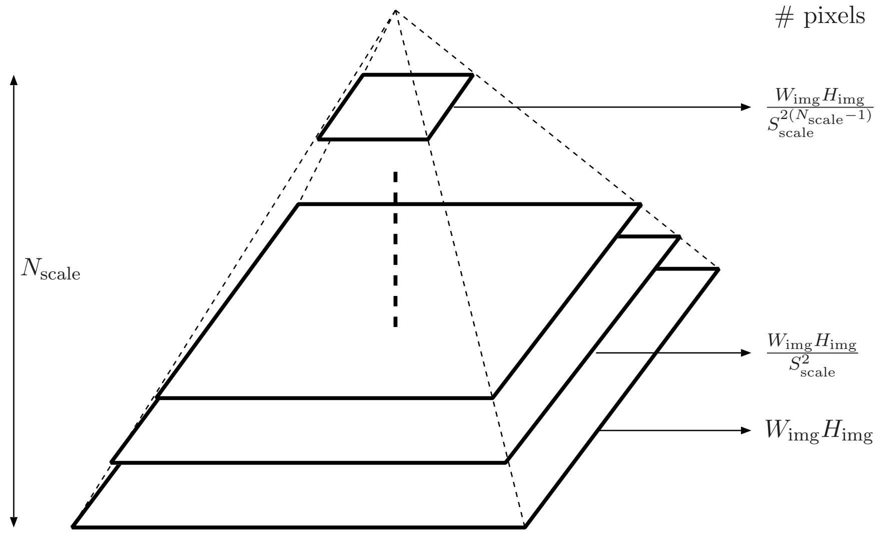
$$N_{\text{pixel}}(w, h) = whN_{\text{ch}} \tag{3.27}$$

where N_{ch} is the number of channels in the image. Then, N_{HOG} is calculated by the summation of the number of required modules for each scaled image:

$$N_{\text{HOG}} = \sum_{i=0}^{N_{\text{scale}}-1} \left\lceil \frac{N_{\text{pixel}}(w_i, h_i)N_{\text{fps}}}{f} \right\rceil \tag{3.28}$$

Table 3.14: Notation for object detection.

Notation	Description
W_{img}	width of input images
H_{img}	height of imput images
N_{ch}	# channels of input images
N_{fps}	frame per second
f	operating frequency
S_{scale}	scale factor
N_{scale}	# scaled images
N_{pixel}	# pixels computed in HOG modules
N_{HOG}	# required HOG modules



# pixels
$\frac{W_{\mathrm{img}}H_{\mathrm{img}}}{S_{\mathrm{scale}}^{2(N_{\mathrm{scale}}-1)}}$
N_{scale}
$\frac{W_{\mathrm{img}}H_{\mathrm{img}}}{S_{\mathrm{scale}}^{2}}$
$W_{\mathrm{img}}H_{\mathrm{img}}$

Figure 3.11: The number of pixels for multi-scale object detection.

Table 3.15: Estimation of LUT utilization for $f = 100$ MHz, color image on FPGA implementation.

Video stream	N_{scale}	S_{scale}	N_{HOG}	LUT utilization	
				DV-HOG	InterHOG [24]
480p30	3	2	37	2.59%	19.24%
1080p30	4	2	249	17.43%	129.48%
1080p60	4	2	498	34.86%	258.96%

where w_i and h_i defined in the following equations are the width and height of a scaled image

$$w_i = \frac{W_{\text{img}}}{(S_{\text{scale}})^i}, \quad h_i = \frac{H_{\text{img}}}{(S_{\text{scale}})^i}, \tag{3.29}$$

S_{scale} is the scale factor of the scaler, and N_{scale} is the number of scaled images as shown in Figure 3.11. Table 3.14 summarizes the notations.

Table 3.15 shows LUT utilization in FPGA implementations estimated for various video streams. The estimation indicates that there is a limitation regarding the LUT utilization on InterHOG adoption for high-resolution video streams. For example, it is impossible to apply a detection system with InterHOG for 1080p30 streams. Considering that video resolutions and frame rates are increasing nowadays, the cell area of a single HOG module is becoming a more critical concern. From the result that DV-HOG occupies only 34.86% LUT even for 1080p60 video streams, DV-HOG is a promising HOG method to use in hardware systems.

3.5 Summary

The primary concern of many studies on visual object detection is how to implement an existing visual object detection algorithm in hardware without accuracy degradation. Since most of the existing algorithms are not hardware oriented, their detection performance has been spoiled by the simplifications introduced in their hardware implementations.

As a solution to this issue, this chapter proposed a hardware-oriented HOG-based feature extraction method, which is called DV-HOG. The proposed DV-HOG feature descriptor can be calculated with low computational complexity while it has equivalent or better representation capability compared with the conventional HOG feature descriptors. Also, DV-HOG achieved stable detection accuracy for fixed-point calculation, and its circuit scale is close to that of a highly simplified implementation of ChenHOG.

Chapter 4

Aggressive Quantization Method for Boosted Decision Trees

This chapter explains an aggressive quantization method for a BDT classifier to reduce memory requirement [65]. First, a preliminary evaluation compares three conventional machine learning algorithms in terms of the influence of numerical precision on classification accuracy. Then, based on the result, this chapter proposes a quantization method for a BDT classifier which focuses on the range of threshold values. The evaluation result shows the relationship between the detection accuracy and numerical precision.

4.1 Introduction

Two essential issues of machine learning are representation capability and efficiency. In terms of representation capability, it had been proved that an MLP [53], an early neural network, is a universal approximator and enough to approximate any complicated target function [66]. The problem is a representational inefficiency that the MLP requires infeasible computational resources to approximate the target function. The recent machine learning algorithms successfully reduced the required computational resources to a feasible level, but it does not change the fact that they require a massive amount of computational cost.

To address this issue, multiple researches have explored the numerical precision required for the training process [67, 68] because the amount of computational resources depends on numerical precision. For accelerating training, various methods based on hardware have been proposed such as NVIDIA's GPU acceleration, IBM's SyNAPSE [69], Manchester University's SpiNNaker [70], and Google's Tensor Processing Unit [1]. The consensus of these researches is that the training process of

[1]https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html

Table 4.1: Format of IEEE 754 double-precision floating point.

Bit range	63	62–52	51–0
Representation	s	e	m
Description	sign	exponent	mantissa

neural networks requires at least half-precision floating point or 16-bit wide fixed point representation. However, from the perspective of embedded systems using machine learning techniques for detection or classification, this issue is not thoroughly discussed at all because no research clarifies necessary and sufficient numerical precision for the classification process or the impact the numerical precision on the classification accuracy of each machine learning algorithm.

This chapter analyzes the classification accuracy under numerical precision changes on three conventional machine learning algorithms: an MLP [53], an SVM [29], and an AdaBoost [30], and elucidates which machine learning algorithm is suitable for embedded systems. Also, based on the analysis, this chapter proposes an aggressive approximation method that can be used for practical applications of embedded systems and evaluates it on ACF object detection [36].

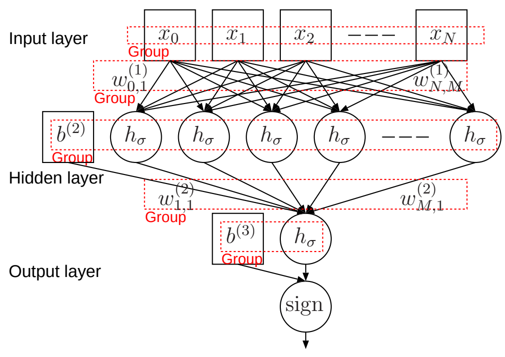
The rest of this chapter is organized as follows. Section 4.2 analyzes the relationship between numerical precision and classification accuracy on an MLP, an SVM, and an AdaBoost. Section 4.3 presents the proposed aggressive approximation method for embedded systems and Section 4.4 evaluates the proposed method on pedestrian detection benchmark. Finally, Section 4.5 concludes this chapter.

4.2 Influence of Numerical Precision on Classification Accuracy

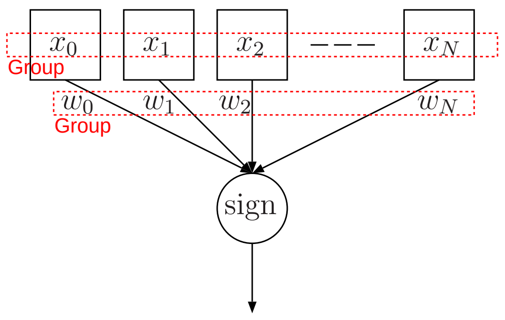
This section compares three types of machine learning algorithms, MLP, SVM, and AdaBoost, in terms of the representation capability in a fixed-point representation in order to make clear which machine learning algorithm is suited for embedded systems. Many hardware implementations use fixed-point representations because hardware architectures of fixed-point require less resources than those of floating-point.

To examine the classification accuracy with limited numerical precision, each machine learning algorithm is modified to constrain the bit width of each parameter. The format of IEEE 754 double-precision floating point [75] consists of three parts: sign s, exponent e, and mantissa m as shown in Table 4.1, and a floating point value v_{flt} can be represented as

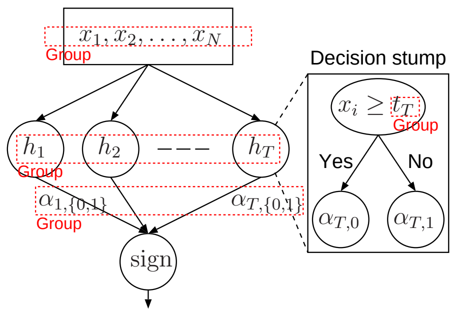
$$v_{\mathrm{flt}} = (-1)^s \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) 2^{e-1023}, \tag{4.1}$$



(a) Multilayer perceptron (MLP).



(b) Support vector machine (SVM).



(c) AdaBoost.

Figure 4.1: Basic structure of machine learning algorithms.

where b_i represents the i-th bit. In this analysis, sets of IEEE 754 double-precision floating-point parameters are converted into sets of a pseudo n-bit fixed-point parameters, which is a floating point that can be represented by an n-bit fixed point. Each set of parameters is surrounded by a dotted box labeled with "Group" in Figure 4.1. A floating point v_{flt} of each set is converted into a pseudo fixed point v_{fix} as follows:

$$v_{\text{fix}} = \frac{\lfloor v_{\text{flt}} \cdot f_{\text{scale}} \rfloor}{f_{\text{scale}}}, \tag{4.2}$$

$$f_{\text{scale}} = 2^{n-(e_{\max}-1023)-2}, \tag{4.3}$$

$$e_{\max} = \max_j e_j, \tag{4.4}$$

where e_j is the exponent of the j-th floating-point value.

For the evaluation, both floating-point and fixed-point machine learning algorithms are implemented based on the OpenCV library [2] and parameters shown in Table 4.2. Each implementation is evaluated on Gisette [73] and Madelon [74] datasets from LIBSVM dataset [76]. Gisette is a two-class handwritten digit recognition dataset of '4' and '9,' which are confusing digits, and this dataset is generated from a well-known digit

[2]`https://opencv.org`

Table 4.2: Parameters and settings of each machine learning algorithm.

Algorithm	Parameter	Setting
MLP	# nodes in hidden layer	Gisette: 100, Madelon: 32
	Learning algorithm	RPROP [71]
SVM	Kernel	Linear
	Learning algorithm	Quadratic programming
AdaBoost	# weak learners	100
	Learning algorithm	Real AdaBoost [72]

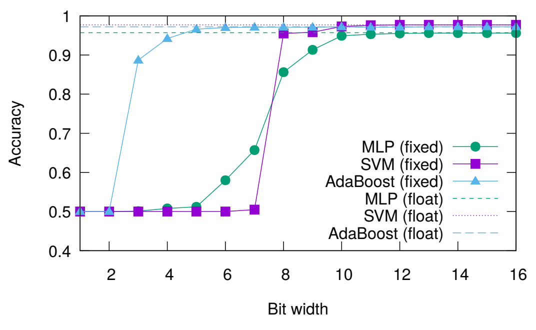
Table 4.3: Details of classification datasets.

Dataset	# of features	# of training data	# of test data
Gisette [73]	5,000	6,000	1,000
Madelon [74]	500	2,000	6,000

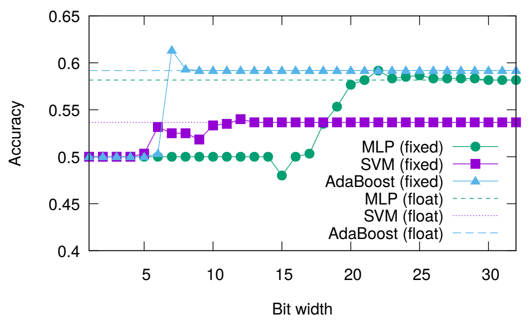
dataset, MNIST [77]. Madelon is an artificially generated non-linear dataset. Table 4.3 describes the quantitative details of both datasets.

Figures 4.2(a) and 4.2(b) show the evaluation results, where the initial accuracies at 1-bit width are decided only by the number of positive and negative data because 1-bit width represents only the sign bit and has no representation capability. In both figures, as the bit width increases, each fixed-point implementation converged to the accuracy of the corresponding floating-point implementation. Especially, the fixed-point AdaBoost achieved equivalent accuracy with a smaller bit width than others: 5-bit and 8-bit widths in Figures 4.2(a) and 4.2(b), respectively.

This result can be explained by the fact that each machine learning algorithm requires a different value range for classification. As shown in Figure 4.3(a), AdaBoost stores threshold values and the range of threshold values is highly limited. On the

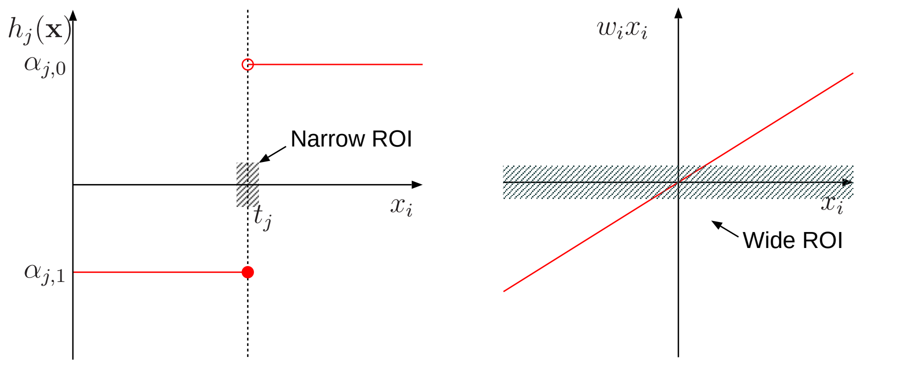


(a) Gisette dataset.



(b) Madelon dataset.

Figure 4.2: Experimental results of classification accuracy with limited numerical precision.



(a) Decision stump $h_j(\mathbf{x})$ of AdaBoost. (b) Inner product $w_i x_i$ of SVM and MLP.

Figure 4.3: Region of interest (ROI) comparison.

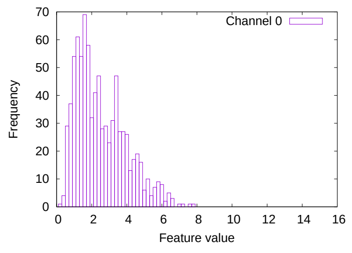
Table 4.4: Range of non-zero frequency of a pedestrian classifier for ACF.

Feature type	Channel #	Threshold range	Feature range	Occupancy (%)
HOG	0	0.266–7.71	0–14.4	51.9
	1	0.346–5.21	0–12.3	39.4
	2	0.244–3.94	0–12.7	29.1
	3	0.159–5.09	0–14.7	33.6
	4	0.179–4.37	0–12.6	33.3
	5	0.276–4.39	0–12	34.2
Magnitude	6	5.92–14	0–17.7	45.7
LUV	7	1.84–5.34	0–5.94	59.0
	8	4.76–6.95	2.19–14.9	17.2
	9	7.05–9.62	1.69–13.9	21.1

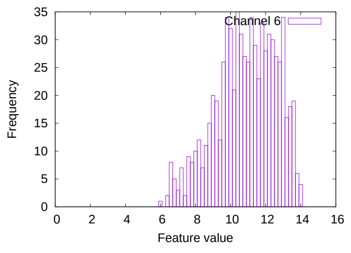
contrary, in MLP and SVM using inner production, the weight values spread widely as shown in Figure 4.3(b).

4.3 Aggressive Approximation for Fixed-Point AdaBoost

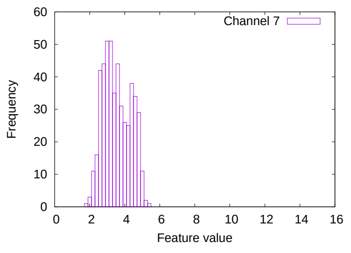
Based on the analysis that AdaBoost is less sensitive to limited numerical precision, this section proposes an aggressive approximation method for the AdaBoost. The proposed aggressive approximation method mainly focuses on the narrow value range around thresholds used in BDT and quantizes the threshold values with small bits. To apply the proposed method into an ACF classifier, it is necessary to examine thresholds of a floating-point ACF classifier in advance. As mentioned in Chapter 2, the ACF classifier for pedestrian detection uses ten aggregated channels, and Table 4.4 shows the value



(a) HOG (Ch. 0).



(b) Magnitude (Ch. 6).



(c) L component of LUV (Ch. 7).

Figure 4.4: Histograms of thresholds.

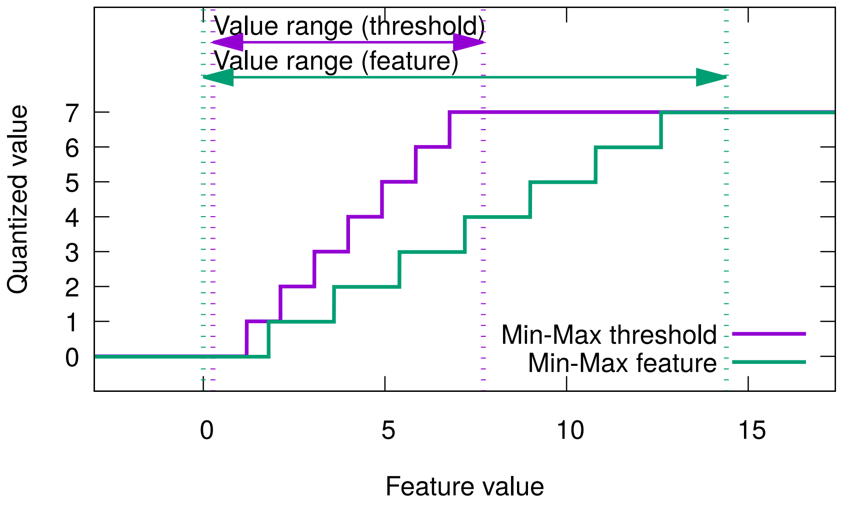


Figure 4.5: 3-bit quantization of features on ACF channel 0.

range of each channel, where threshold and feature ranges represent the value ranges of thresholds and entire features, respectively. As shown in Table 4.4, the threshold ranges occupy only 17.2% to 59.0% of the feature ranges. Moreover, as shown in Figure 4.4, each distribution shows a mountain-shaped graph, and it represents the possibility that the middle of distribution is more important than both ends of distribution for classification.

To verify this hypothesis, two types of approximation methods are examined: one is to quantize features based on feature ranges, and the other is to quantize features based on threshold ranges. Both quantizations are defined as follows:

$$v_{\mathrm{fix}} = \left\lfloor 2^n \cdot \frac{\min\left(r_{\max}, \max\left(r_{\min}, v_{\mathrm{flt}}\right)\right) - r_{\min}}{r_{\max} - r_{\min} + \epsilon} \right\rfloor, \tag{4.5}$$

where $r_{\min}$ and $r_{\max}$ respectively represent the minimum and the maximum of the range and ϵ represents a small positive infinitesimal quantity. For the convenience of

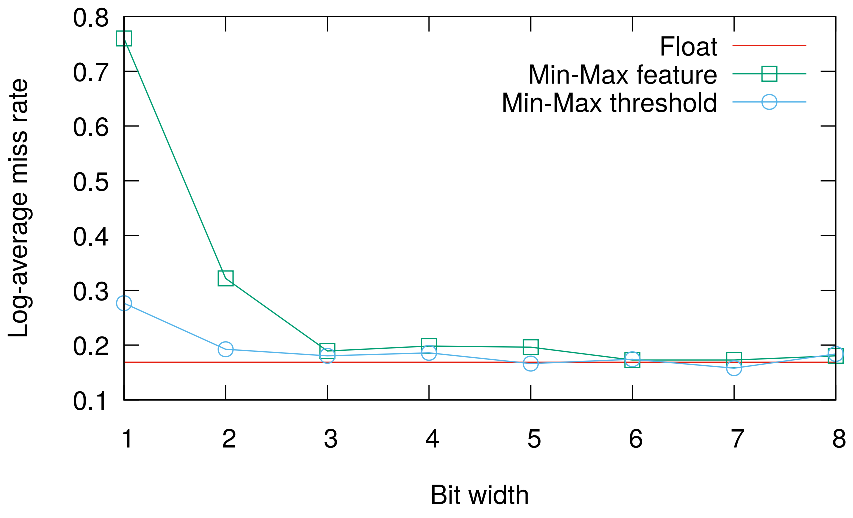


Figure 4.6: Comparison of classification performances.

explanation, both methods are referred to as *Min-Max feature* and *Min-Max threshold*, respectively. An example of each quantization method is described in Figure 4.5 using channel 0 of the aggregated channel. As shown in Figure 4.5, Min-Max threshold has a higher resolution around the thresholds than Min-Max feature and ignores the outside of the region, which does not affect the comparison result.

4.4 Evaluation on Pedestrian Detection

This section evaluates the proposed method on an ACF classifier used in practical applications. The evaluation uses INRIA Person Dataset [24] and Caltech Pedestrian Detection Benchmark [25]. To show the influence of numerical precision, Figure 4.6 plots the change of log-average MR. The proposed Min-Max threshold showed better classification accuracy than the Min-Max feature and achieved 19.2% log-average miss rate at only 2-bit width; the accuracy 19.2% is only 2% worse than the original ACF classifier. For a detailed comparison, Figure 4.7 shows DET curves at 2-bit width. As shown in Figure 4.7, the proposed method outperforms the Min-Max feature and achieves almost equivalent accuracy to the original ACF classifier over the entire range of false positive per image. Also, the classification accuracy can be confirmed from actual detection results shown in Figures 4.8, 4.9, and 4.10, in which false positives and false negatives are labeled with “FP” and “FN” respectively. The proposed method’s insensitivity to numerical precision can be useful for both software and hardware implementations. The reduction of bit width reduces memory size, circuit area, and power consumption, which is significantly helpful in embedded systems. For instance, as shown in Table 4.5, pedestrian detection using a Full HD image requires 39.6 Mbit

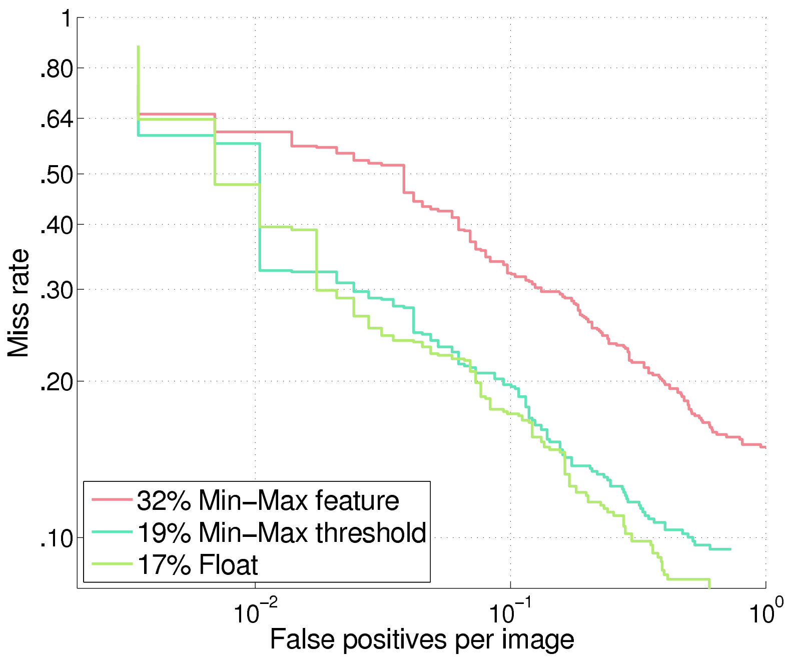


Figure 4.7: Comparison of DET curves.

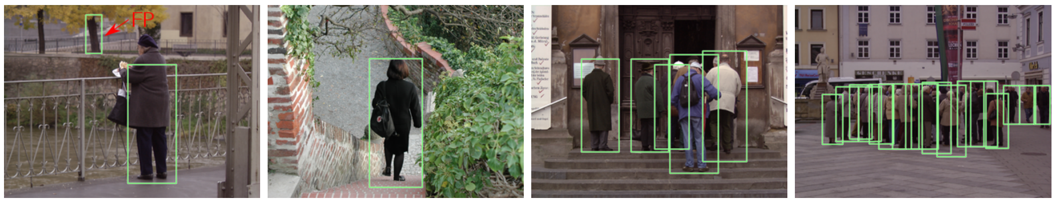
Table 4.5: Memory usage comparison for a Full HD image.

Method	Representation	Bit width	Memory usage
ACF [36]	Floating point	32	39.6 Mbit
Proposed	Fixed point	2	2.47 Mbit

and 2.47 Mbit in the original ACF and the proposed method respectively, and this memory usage directly affects the computational cost for implementation.

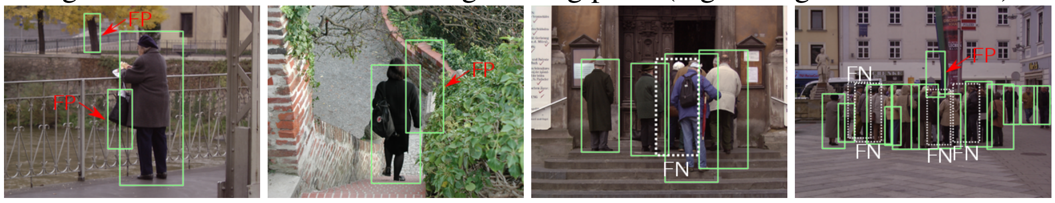
4.5 Summary

This chapter analyzed the influence of numerical precision on the classification accuracy of three representative machine learning algorithms and clarified that the AdaBoost is less sensitive to limited numerical precision due to the narrow value range required for classification. Also, even though it is a common practice to use fixed-point representation in embedded systems, the proposed method based on the analysis achieved remarkable bit-width reduction with a slight degradation of classification accuracy: the proposed method reduced memory requirements to a one-sixteenth level required by the ACF classifier in a test case. So far, it has been difficult to realize novel machine learning algorithms on embedded systems due to the severe computational resource requirement. However, the proposed method makes it possible to significantly



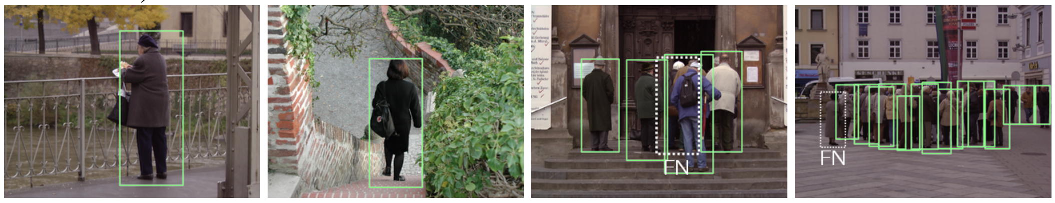
(a) person_056.png. (b) person_076.png. (c) person_190.png. (d) person_200.png.

Figure 4.8: Detection results using floating point (log-average MR = 16.9%).



(a) person_056.png. (b) person_076.png. (c) person_190.png. (d) person_200.png.

Figure 4.9: Detection results using 2-bit wide fixed point (Min-Max feature, log-average MR = 32.2%).



(a) person_056.png. (b) person_076.png. (c) person_190.png. (d) person_200.png.

Figure 4.10: Detection results using 2-bit wide fixed point (Min-Max threshold, log-average MR = 19.2%).

reduce the computational resources required by the algorithms, so that the proposed method can contribute to popularizing embedded systems for machine learning. In this chapter, the proposed method focused only on the computational resource of the classification process, but the experimental result also implies the possibility that feature extraction can be aggressively approximated because the AdaBoost classification does not require high numerical precision.

Chapter 5

Hardware Architecture for Parallel Boosted Decision Trees Classification

This chapter proposes a hardware architecture for parallel BDT classification [78]. The proposed hardware architecture utilizes the parallelism for three dimensions for speed-up like a SIMD manner. Also, to avoid memory access conflict originating from parallel classification, a task scheduling algorithm that fixes the memory access order is proposed. The evaluation on FPGA implementation shows that the proposed hardware architecture of 1,024 parallel classifications is implementable on FPGA and it contributes to speed up.

5.1 Introduction

As mentioned in Chapter 1, object detection methods using BDT have multiple advantages in hardware implementation: low computational cost with soft cascade and multiplier-free operations in classification. Even with these attractive features, only a few studies on decision tree hardware architectures are reported [40], because conditional branches of decision stumps composing BDT require random memory accesses, which prevents efficient parallel processing.

As a solution, this chapter proposes a hardware architecture for BDT. The proposed architecture processes object detection in a SIMD-like homomorphic manner even while BDT are adopted as a classifier. The proposed architecture has two following distinctive features, which improves processing performance. First, it supports three-dimensional parallel memory access, 1-D for feature channels and 2-D for image space, achieving multiple times higher processing performance than conventional hardware architectures. Second, it takes advantage of algorithmic acceleration by using soft cascade, which improves processing performance by over one to two orders of magnitude. Then, this chapter proposes a task scheduling algorithm to control memory accesses from multiple modules to avoid memory access conflict. For evaluating the proposed BDT

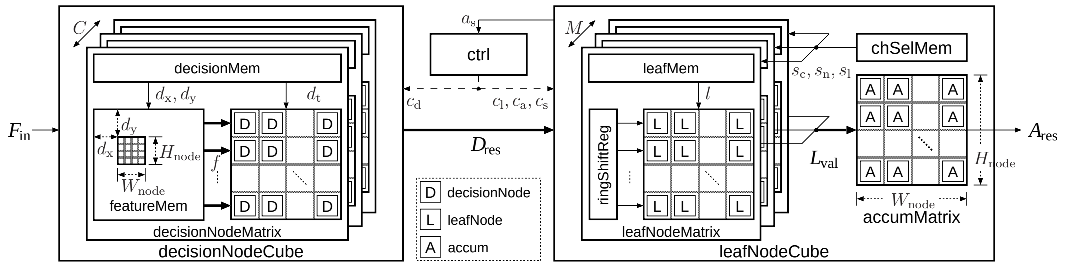


Figure 5.1: Hardware architecture overview.

hardware architecture, ACF [36] features are assumed, and classification using multi-scale classifiers with octave-wise feature maps [37] are used.

The rest of this chapter is organized as follows. Section 5.2 provides a hardware architecture for BDT, and Section 5.3 proposes a task scheduling algorithm to resolve memory access conflict. Section 5.4 describes the evaluation results and provides an analysis of the proposed hardware architecture. Section 5.5 concludes the chapter.

5.2 Parallel Implementation of Boosted Decision Trees using Multiple Memory Banks

The proposed hardware architecture is a single-node architecture, which is explained in Section 2.4.1 devised for exploiting multiple memory banks with a small amount of routing resources. This section explains its overview and details in order.

5.2.1 Architecture Overview

ACF has multiple types of features and requires sophisticated memory access patterns. Considering the parallel feature extraction before classification, it is necessary to allocate a dedicated memory bank for each channel. In this case, the threshold network and the single-path architecture explained in Section 2.4 are not suitable because they require a massive amount of routing resources for supporting random memory access to all the banks. On the other hand, the single-node architecture can resolve the routing resource problem by assigning a universal node to each channel and merging the responses of decision nodes belonging to each DT.

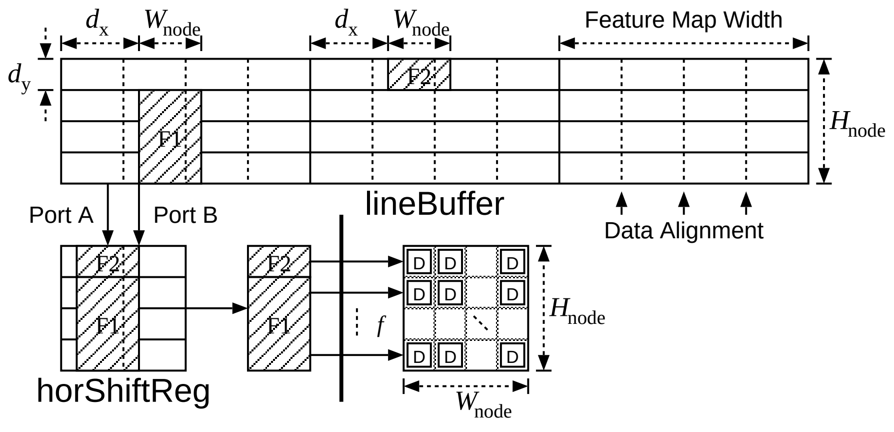
The proposed architecture is designed based on the idea mentioned above. Figure 5.1 shows an overview of the proposed hardware architecture, and Table 5.1 lists the notations used in Figure 5.1. The proposed architecture mainly consists of three sub-modules: `decisionNodeCube`, `leafNodeCube`, and `ctrl`, where they are a 3-D array of decision nodes, a 3-D array of leaf nodes, and a control unit, respectively. The `decisionNodeCube` consists of C `decisionNodeMatrix` modules,

Table 5.1: Notation and description of each module.

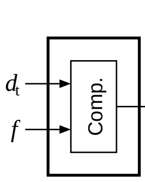
Module	Notation	Description
Top Module	F_{in}	A set of input features
	D_{res}	A set of decision responses
	L_{val}	A set of leaf values
	A_{res}	A set of accumulated responses
	W_{node}	#horizontal D/L/A nodes of a matrix
	H_{node}	#vertical D/L/A nodes of a matrix
	C	#decision node matrices
	M	#leaf node matrices
`decisionMem`	d_{x}	Feature x position
	d_{y}	Feature y position
	d_{t}	Decision node threshold
`featureMem`	f	All feature values of a block
`leafMem`	l	All leaf values of a DT
`accumMatrix`	a_{s}	Sign bit for soft cascade
`chSelMem`	s_{c}	Channel index
	s_{n}	Node index
	s_{l}	Last node flag
`ctrl`	c_{s}	Address control signal
	c_{d}	`decisionNodeCube` control signal
	c_{l}	`leafNodeCube` control signal
	c_{a}	`accumMatrix` control signal
Misc.	d_{res}	A decision response
	l_{val}	A leaf value
	a_{res}	An accumulated response

and the `leafNodeCube` consists of M `leafNodeMatrix` modules, an `accumMatrix` module, and a `chSelMem` module, where M is less or equal to C due to the non-uniformity of the channel usage described in Section 5.3. The `decisionNodeCube` receives feature input F_{in} and outputs decision responses D_{res}. The `leafNodeCube` selects corresponding leaf values L_{val} from a part of the decision responses D_{res} and accumulates the leaf values to calculate the final responses A_{res}.

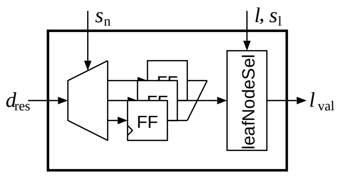
This architecture enables a 3-D parallel classification, and the hardware handles a massive amount of data. Thus, in hardware design, the scalability of the architecture for each sub-module, `decisionNodeMatrix`, `leafNodeMatrix`, and `accumMatrix` needs to be carefully considered, which will be discussed in Section 5.2.2.



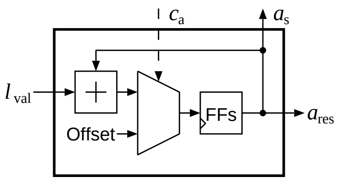
(a) featureMem.



(b) decisionNode.



(c) leafNode.



(d) accum.

Figure 5.2: Block diagrams of processing elements.

5.2.2 Details of Sub-modules

In the proposed architecture, its processing flow completely depends on the `ctrl` and the `chSelMem` modules. The `ctrl` observes the states of all the sub-modules and dynamically provides control signals, and the `chSelMem` provides static task schedules generated by the proposed scheduling algorithm described in Section 5.3. Therefore, the proposed architecture can handle any BDT by updating task schedules.

Each of C `decisionNodeMatrix` modules composing the `decisionNodeCube` corresponds to one of C input feature channels: HOG channels, LUV channels, and a gradient magnitude channel described in Section 2.2. Each `decisionNodeMatrix` consists of three sub-modules: a `decisionMem`, a `featureMem`, and a 2-D array of $W_{\mathrm{node}} \times H_{\mathrm{node}}$ `decisionNode` modules. In the `decisionNodeMatrix`, the `decisionMem` is the only module controlled by the signal c_{d} from the control unit, and the data, d_{x}, d_{y}, and d_{t}, loaded from `decisionMem` controls the others. The `featureMem` provides $W_{\mathrm{node}} \times H_{\mathrm{node}}$ feature values, f, of the block at the $(d_{\mathrm{x}}, d_{\mathrm{y}})$ position to `decisionNode` modules. To support loading the feature block at an arbitrary position, `featureMem` uses H_{node} dual-port line buffers, `lineBuffer`, and H_{node} shift registers, `horShiftReg`, for location adjustment as shown in Figure 5.2(a). The dual-port line buffers enable to load malaligned data at any vertical position in a single cycle, and shift registers enable to extract the target data columns at any horizontal position.

Each `decisionNode` generates a 1-bit comparison result as the decision response between a feature value of f and a threshold d_{t} as shown in Figure 5.2(b), where d_{t} is the threshold shared in all `decisionNode` modules of a `decisionNodeMatrix`.

Each of M `leafNodeMatrix` modules composing the `leafNodeCube` consists of three sub-modules: a `leafMem`, a `ringShiftReg`, and a 2-D array of $W_{\mathrm{node}} \times H_{\mathrm{node}}$ `leafNode` modules. The `leafMem` provides all the leaf values of each DT to `leafNodeMatrix`, the `ringShiftReg` vertically and horizontally rotates the $W_{\mathrm{node}} \times H_{\mathrm{node}} \times C$ 1-bit decision responses to correct positions, and `leafNode` selects a leaf value from l based on each series of decision responses. Figure 5.2(c) shows the details of the `leafNode` for depth-two BDT. As shown in Figure 5.2(c), the `leafNode` includes a demultiplexer for rearranging the order of decision responses, three flip-flops (FFs) for storing the decision responses of a depth-two DT, a `leafNodeSel` for selecting a leaf value based on the responses. This structure enables the `leafNode` to handle the random input order of decision responses and to improve the processing performance by task scheduling technique. Also, a simple modification of the `leafNode` allows to handle BDT deeper than depth-two BDT: for processing depth-three BDT, the `leafNode` requires 7-bit FFs to store seven decision responses, and the `leafNodeSel` requires an extension to select a leaf value of eight leaf values.

The `accumMatrix` consists of $W_{\mathrm{node}} \times H_{\mathrm{node}}$ `accum` modules. Figure 5.2(d) describes the details of the `accum`. The `accum` accumulates the leaf values of each BDT by using an adder and FFs, where the FFs are initialized with the offset, and the offset is used for soft cascade rejection. When the static threshold of the soft cascade is $-s$, the offset is set to s, so that the control unit can decide soft cascade rejection only with the sign bit of the accumulated value, a_{s}.

5.3 Task Scheduling for Parallel Implementation

The proposed hardware architecture can process decision nodes of a DT in arbitrary order, and its processing performance depends on the efficiency of the parallel memory access. For further acceleration, this section proposes a task scheduling algorithm dedicated to the proposed architecture. This task scheduling is an optimization problem considering each decision node as a task subject to constraints derived from the architecture design. This section explains how to formulate the task scheduling problem, describes the proposed algorithm, and analyzes its effectiveness.

5.3.1 Boosted Decision Trees Scheduling Problem

Memory accesses resulting from conditional branches may cause memory conflict when processing multiple DTs at once. The purpose of task scheduling is to avoid this memory conflict by processing all the decision nodes of a DT in a fixed order as shown in Figure 5.3 and controlling parallel memory accesses from multiple DTs. Given a BDT

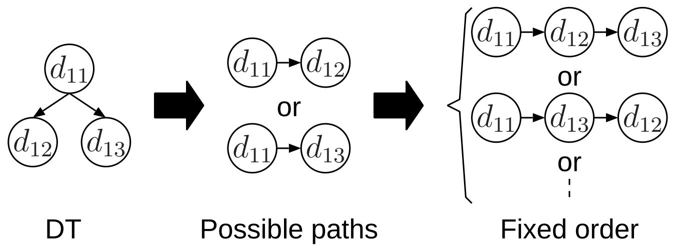


Figure 5.3: Input dependency removal of DT by visiting all nodes.

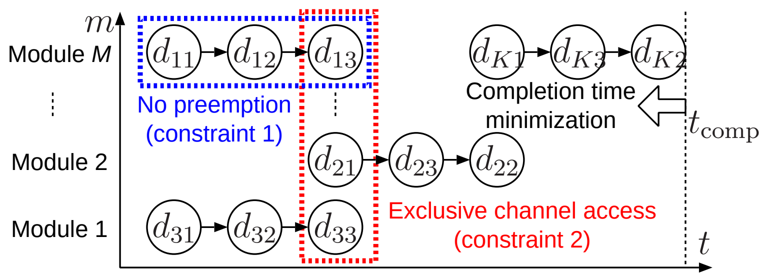

m
Module M
Module 2
Module 1
No preemption (constraint 1)
Exclusive channel access (constraint 2)
Completion time minimization
t_{comp}
t
d_{11} d_{12} d_{13}
d_{21} d_{23} d_{22}
d_{31} d_{32} d_{33}
d_{K1} d_{K3} d_{K2}

Figure 5.4: Target scheduling problem.

classifier, this scheduling algorithm determines a task schedule in an offline manner, and classification requires no additional computation for scheduling. The task scheduling is an optimization problem finding the minimum completion time t^*_{comp} and its assignment matrix A^* for M modules defined as

$$t^*_{\mathrm{comp}} := \min t_{\mathrm{comp}}(A), \quad A^* := \operatorname*{argmin}_{A} t_{\mathrm{comp}}(A), \tag{5.1}$$

where $t_{\mathrm{comp}}(A) = \max\{t \mid \exists m \in \{1, \dots, M\}, a_{mt} \neq 0\}$, in which $t_{\mathrm{comp}}(A)$ is the completion time using an assignment matrix A, and a_{mt} is the (m, t)-th entry of A representing the decision node processed on the m-th module at the t-th cycle. If there is no task assignment, a_{mt} will be zero. Figure 5.4 shows two constraints for task scheduling derived from the hardware architecture. The first constraint is that each `leafNode` needs to process all the decision nodes belonging to a DT in consecutive cycles. This is because each response to a decision node is not shared between modules, and there is no room for storing responses of multiple DTs. The second constraint is that each `leafNodeMatrix` exclusively uses decision responses from a `decisionNodeMatrix` at each time to avoid memory access conflict. The two constraints of this scheduling problem are defined as follows. When a decision node d_{ks}, which is the s-th decision node of the k-th DT, is assigned to the m_{ks}-th module at the t_{ks} cycle, i.e., $a_{m_{ks}t_{ks}} = d_{ks}$, the first constraint is

$$\forall k \in \{1, \dots, K\}, \forall s_1, s_2 \in \{1, \dots, S\}, m_{ks_1} = m_{ks_2}, \tag{5.2}$$

Algorithm 5.1 Task scheduling of BDT.

```
Input:
    $H := \{h_k \mid h_k = \{d_{k1}, \dots, d_{kS}\}, 1 \leq k \leq K\}$,
    $M :=$ parallel degree
Output:
    $A =$ Assignment matrix
    $t^*_{\mathrm{comp}} =$ completion time of $A$
 1: $c \leftarrow \textsc{SortAndMergeChannels}(H)$
 2: $T_{\max} \leftarrow ST$
 3: $A \leftarrow O \in \mathbb{N}_+^{M \times T_{\max}}$
 4: for $n = 1$ to $M$ do
 5:     $H_n \leftarrow \{h_n \mid \exists j, c(d_{ks}) = n\}$
 6:     for all $h \in H_n$ do
 7:         $(m^*, t^*_{\mathrm{tgt}}, t^*) \leftarrow (0, T_{\max}, T_{\max})$
 8:         $P(h)$: a set of tuples consisting of permutation of $h$
 9:         for all $P \in P(h)$ do
10:             $(m, t) \leftarrow \textsc{SearchWithConstraint}(A, P)$
11:             $t_{\mathrm{tgt}} \leftarrow t + \max\{i \mid c(p_i) = n\}$
12:             if $((t_{\mathrm{tgt}} < t^*_{\mathrm{tgt}}) \vee ((t_{\mathrm{tgt}} = t^*_{\mathrm{tgt}}) \wedge (t < t^*)))$ then
13:                 $(m^*, t^*_{\mathrm{tgt}}, t^*) \leftarrow (m, t_{\mathrm{tgt}}, t)$, $P^* \leftarrow P$
14:             end if
15:         end for
16:         for $s = 1$ to $S$ do
17:             $a_{m^*, t^*+s} \leftarrow p^*_s$
18:         end for
19:     end for
20:     $H \leftarrow H \setminus H_n$
21: end for
22: $t^*_{\mathrm{comp}} \leftarrow \max\{t \mid \exists m \in \{1, \dots, M\}, a_{mt} \neq 0\}$
```

where S is the number of decision nodes in a DT, and for any k, each t_{ks} needs to be a consecutive number. The second constraint is

$$\begin{aligned} &\forall t \in \{1, \dots, T_{\max}\}, \forall m_1, m_2 \in \{1, \dots, M\}, m_1 \neq m_2, \\ &\quad c(a_{m_1 t}) \neq 0, c(a_{m_2 t}) \neq 0, c(a_{m_1 t}) \neq c(a_{m_2 t}), \end{aligned} \tag{5.3}$$

where $c(d_{ks})$ represents the channel used in d_{ks}, and $T_{\max}$ is the possible maximum completion time. Then, the scheduling problem can be defined as an offline problem. This scheduling problem can be considered as an extension of the $\mathcal{NP}$-hard job shop scheduling problem, and then it is $\mathcal{NP}$-hard.

Algorithm 5.2 Procedures used in Algorithm 5.1.

```
1: procedure SortAndMergeChannels($H$)
2:     $C_{\text{hist}} \leftarrow$ calculate channel histogram from $H$
3:     $x \leftarrow |C_{\text{hist}}|$
4:     while $x > M$ do
5:         $(c_1, \ldots, c_{x-1}, c_x) \leftarrow$ sort $C_{\text{hist}}$ in descending order
6:         $C_{\text{hist}} \leftarrow (c_1, \ldots, c_{x-2}, c_{x-1} + c_x)$: merge channels
7:         Update $c(d_{ks})$
8:         $x \leftarrow x - 1$
9:     end while
10:    $(c_1, \ldots, c_{x-1}, c_x) \leftarrow$ sort $C_{\text{hist}}$ in descending order
11:    Update $c(d_{ks})$
12:    return $c$
13: end procedure

14: procedure SearchWithConstraint($A, P$)
15:    $(m^*, t^*) \leftarrow (0, T_{\max} - |P|)$
16:    for $(m, t) \in \{1, \ldots, M\} \times \{1, \ldots, t^*\}$ do
17:        if $(\forall(m', t') \in \{1, \ldots, M\} \times \{1, \ldots, |P|\},$
                 $a_{m,t+t'} = 0 \land c(a_{m',t+t'}) \neq c(p_{t'}))$ then
18:            if $((t < t^*))$ then
19:                $(m^*, t^*) \leftarrow (m, t)$
20:            end if
21:        end if
22:    end for
23:    return $(m^*, t^*)$
24: end procedure
```

5.3.2 Proposed Heuristic Scheduling Algorithm

As mentioned above, the target scheduling problem is $\mathcal{NP}$-hard, and it is difficult to find the optimal solution t^*_{comp}. Then, the proposed algorithm aims to find a solution which is close to the lower bound, where the lower bound is equal to the maximum number of frequency in channel histogram. The proposed algorithm adopts a greedy approach and focuses on the frequency of channels in BDT. The assignment is performed such that the frequency of channels represents the priority, which reduces the number of assignment candidates and reduces the amount of computation. Also, for improving the completion time, it is a promising approach to make a flat histogram by reducing the number of channels considering the variations of the frequency in the channel histogram.

Algorithm 5.1 shows the proposed algorithm. As a preprocessing, the proposed method merges the input C channels to M channels, where the two channels of lowest

Table 5.2: Detailed scheduling result on $M = 8$.

Depth	Lower bound / t_{comp}	Occupancy	c_{neg}
2	936 / 936	82.1%	50.9
3	1,820 / 1,820	80.4%	59.4

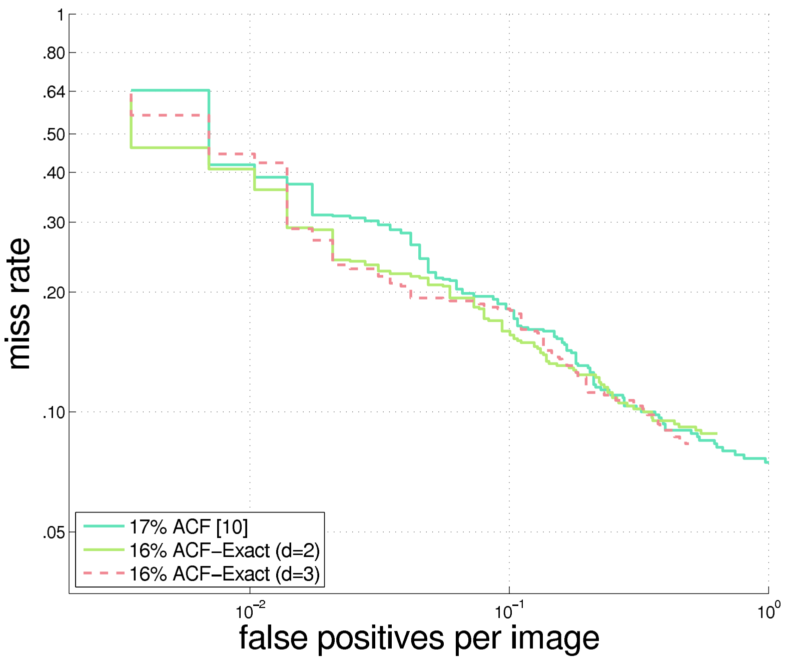
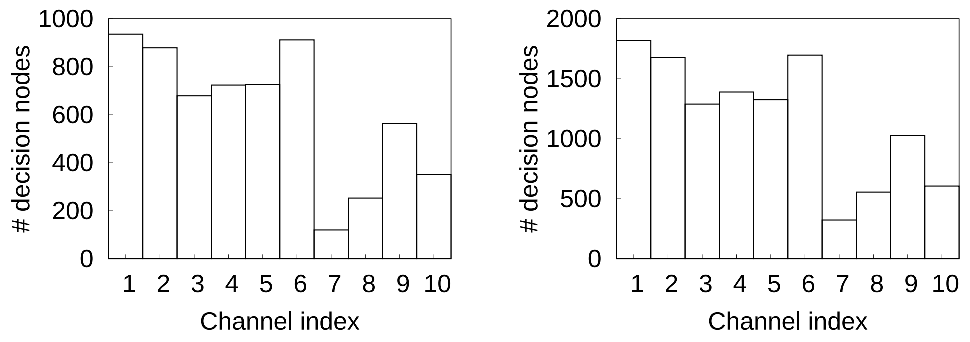


Figure 5.5: Detection error trade-off curves on INRIA Person Dataset.

frequencies are merged in each iteration as described in lines 1–13 of Algorithm 5.2. The assignment process consists of M iterations of the merged channels, and in its n-th iteration, DTs containing the channel n, represented as H_n, are assigned. For each DT h in H_n, the proposed algorithm searches the assignment position satisfying the constraint described in Eqs. (5.2) and (5.3) for all the patterns of processing orders as described in lines 14–24. From all of the processing orders, the one with the earliest completion time and the channel n is selected using the condition in line 12, and it is assigned to an assignment matrix.

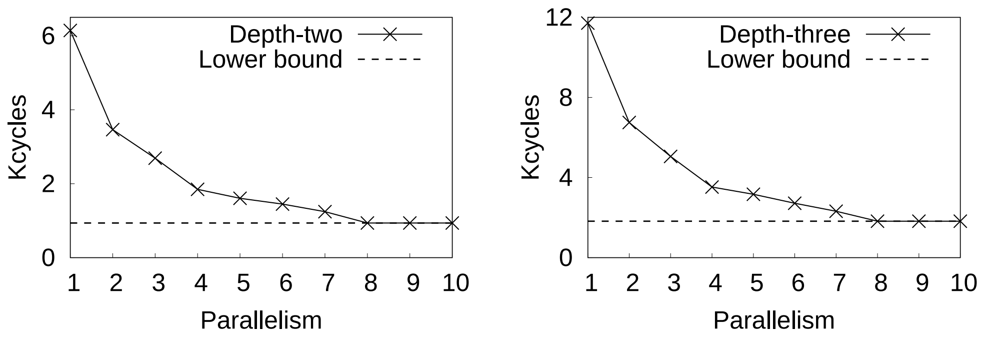
5.3.3 Analysis of Scheduling Algorithm

In the analysis, the target classifiers are depth-two and depth-three ACF classifiers trained in the same manner as [36], consisting of 2,048 and 1,673 DTs. Caltech Pedestrian Detection Benchmark [25] is used to evaluate detection accuracy. The log-average MRs on INRIA Person Dataset [24] are 16.5% and 16.3%, respectively. Figure 5.5 shows the detection error trade-off curves of these two classifiers and the



(a) Depth-two classifier. (b) Depth-three classifier.

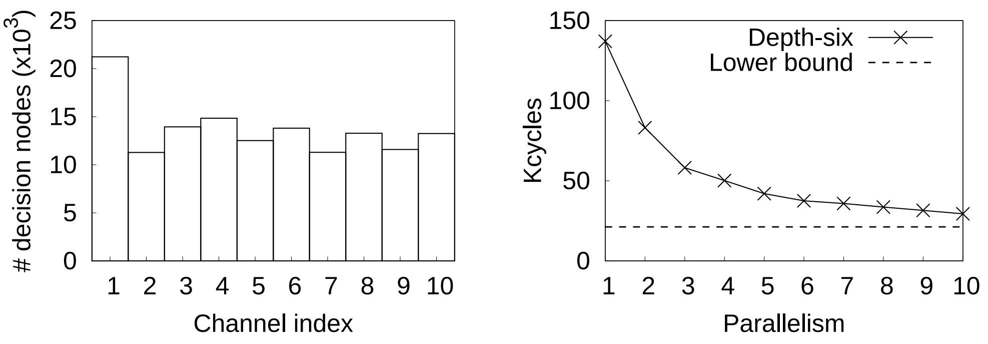
Figure 5.6: Histograms of input channels.



(a) Depth-two classifier. (b) Depth-three classifier.

Figure 5.7: Scheduling results for the different number of modules.

classifier reported in [36], where the classifiers mentioned above achieve equivalent detection accuracy to the original ACF classifier. Figure 5.6 shows the histograms of input channels for these classifiers, which indicates that there exists large variance of frequencies between channels in both histograms. Taking into account the memory access exclusiveness, the lower bound for this problem is equal to the maximum number of decision nodes in a channel. Figure 5.7 shows the relationship between the parallel degree M and the number of the cycles required for processing the classifiers based on the task schedules. For both classifiers, the number of processing cycles decreases as M increases until 8. When M is equal to 8, both numbers of processing cycles reach the lower bound drawn in dotted lines. Compared with serial classification, the proposed scheduling achieves 6.6 and 6.4 times speed up for the depth-two and depth-three classifiers, respectively. For more details, Table 5.2 lists the number of processing cycles and the occupancy of the `leafNodeCube`. The result shows that the proposed scheduling reduces the number of cycles to the lower bound. Also, using soft cascade enables to accelerate the processing performance of negative windows. In Table 5.2,



(a) Histogram of input channels. (b) Scheduling results.

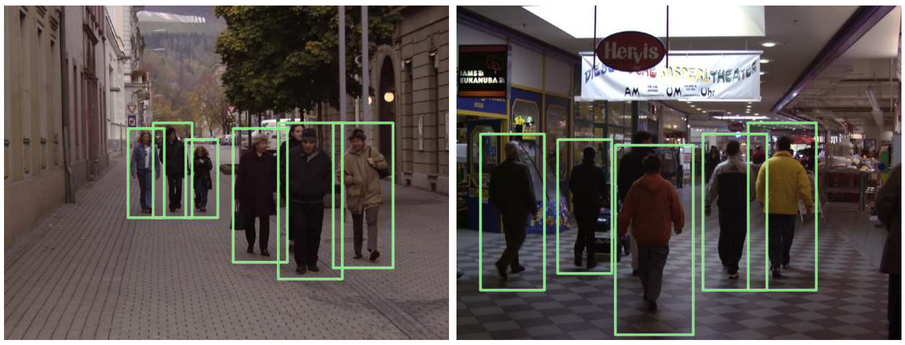
Figure 5.8: Result of depth-six ACF classifier.

c_{neg} represents the average number of processing cycles for negative windows, and Table 5.2 shows that combining the proposed task scheduling and soft cascade can reduce both average cycles of depth-two and depth-three ACF classifiers to 3.3% and 5.4% of processing cycles required for a positive window.

5.3.4 Scheduling under Deeper Boosted Decision Trees

Recent work [79] reports that deeper DTs show good detection performance. To analyze the relationship between the task scheduling performance and the depth of DTs, the proposed task scheduling is applied to a deep BDT classifier. The evaluation uses a depth-six classifier provided by the authors[1], which is trained for Caltech Pedestrian Detection Benchmark [25]. The classifier consists of 3,324 DTs, and the number of decision nodes in the classifier is 137,043. The number of available permutations calculated in line 8 in Algorithm 5.1 is exponentially proportional to the depth of a DT, and then it is necessary to reduce the number of candidates for deep DTs. For mitigating this, the processing order is fixed to the order of channel frequency in this experiment. Figures 5.8(a) and 5.8(b) show its channel histogram and the scheduling results, respectively. The result shows the similar convergence curve to the shallow BDT and achieves 4.1 times speed-up compared with the serial implementation when M is equal to 8. However, the processing cycles do not reach the lower bound even when the parallelism is equal to the channel since the constraint of the exclusive channel access, as in Eq. (5.3), is difficult to satisfy for all the decision nodes. Improving the task scheduling for deeper DTs is included in future work.

[1]`https://eshed1.github.io/code/BoostICPR.zip`



(a) person_191. (b) person_217.

Figure 5.9: Detection results from INRIA Person Dataset.

Table 5.3: FPGA implementation settings.

Target device	Xilinx xc7z045t-2ffg900
Synthesis tool	Vivado 2015.4.2
Simulation tool	ModelSim SE-64 10.3
Target frequency	100MHz
Parallelism	1,024 (channel: 8, block size: 8x16)

5.4 Evaluation

This section explains how to generate a fixed-point classifier and implementation settings, used in the evaluation, and evaluates the FPGA implementation based on the proposed hardware architecture regarding resource usage and processing performance.

5.4.1 Evaluation Settings

For the hardware implementation, the depth-two BDT described in Section 5.3 is converted into a classifier in fixed-point representation by using the method explained in Chapter 4. Figure 5.9 shows the detection results of the converted fixed-point classifier. The proposed hardware architecture is implemented using Verilog HDL at register transfer level (RTL). Table 5.3 shows the implementation settings. The target device is Xilinx xc7z045t-2ffg900, the target operating frequency is 100MHz, and the degree of parallelism is 1,024: parallel degree 8 from feature channels and 128 from image blocks, respectively. Feature extraction uses three types of feature descriptors, i.e., HOG, gradient magnitude, and RGB color channels. The evaluation uses RGB channels instead of LUV channels because the difference of color channels does not cause notable accuracy loss and converting to LUV channels is computationally intensive [62]. Also, for hardware implementation efficiency, the classification procedure proposed by Benenson *et al.* [37] is assumed, which uses multiple classifiers corresponding to

Table 5.4: FPGA resource utilization.

Module	Slice		LUT (Logic)		LUT (Memory)	
`decisionNodeCube`	7,796	(14.3%)	23,891	(10.9%)	0	(0.0%)
→ 10 `featureMem`	5,649	(10.3%)	15,609	(7.1%)	0	(0.0%)
→ 10 `decisionMem`	163	(0.3%)	252	(0.1%)	0	(0.0%)
`leafNodeCube`	10,822	(19.8%)	35,143	(16.1%)	1	(0.0%)
→ 8 `leafMem`	476	(0.9%)	1,114	(0.5%)	0	(0.0%)
→ `chSelMem`	1,090	(2.0%)	2,086	(1.0%)	0	(0.0%)
→ `accumMatrix`	1,952	(3.6%)	5,541	(2.5%)	0	(0.0%)
`ctrl`	286	(0.7%)	503	(0.2%)	0	(0.0%)
→ `decisionNodeCtrl`	194	(0.4%)	396	(0.2%)	0	(0.0%)
→ `leafNodeCtrl`	92	(0.2%)	107	(0.0%)	0	(0.0%)
Total	18,904	(34.6%)	59,537	(27.2%)	1	(0.0%)

Module	LUT (FF)		32Kb BRAM		DSP	
`decisionNodeCube`	24,420	(11.2%)	170	(31.2%)	0	(0.0%)
→ 10 `featureMem`	15,602	(7.1%)	160	(29.4%)	0	(0.0%)
→ 10 `decisionMem`	262	(0.1%)	10	(1.8%)	0	(0.0%)
`leafNodeCube`	36,839	(16.9%)	16	(2.9%)	0	(0.0%)
→ 8 `leafMem`	1,125	(0.5%)	8	(1.5%)	0	(0.0%)
→ `chSelMem`	2,086	(1.0%)	8	(1.5%)	0	(0.0%)
→ `accumMatrix`	5,557	(2.5%)	0	(0.0%)	0	(0.0%)
`ctrl`	420	(0.2%)	0	(0.0%)	1	(0.1%)
→ `decisionNodeCtrl`	315	(0.1%)	0	(0.0%)	1	(0.1%)
→ `leafNodeCtrl`	105	(0.0%)	0	(0.0%)	0	(0.0%)
Total	61,679	(28.2%)	186	(34.1%)	1	(0.1%)

different window sizes and feature maps extracted from scaled images, instead of the genuine ACF classification procedure using a classifier and FFP proposed in [36]. Although FFP shows efficient memory usage and higher processing performance for software implementation, it is not suitable for hardware implementation because FFP needs to generate each layer of feature pyramid sequentially.

5.4.2 Resource Usage

For the evaluation of resource utilization, the RTL implementation is synthesized with Vivado 2015.4.2. Table 5.4 shows the resource utilization of the proposed implementation. As in Table 5.4, it occupies less than 35% of both slice and block RAM resources of the target FPGA for processing 1,024 decision nodes in parallel. Also, from the details of LUT usage, the balanced use of both LUTs and FFs can be confirmed. Therefore, the proposed BDT hardware architecture is suitable for the object detection system explained in Chapter 6.

5.5 Summary

For practical applications using visual object detection, the improvement of the trade-offs between hardware resources, processing performance, and detection accuracy has been a critical issue, and the proposed architecture successfully resolved this issue by improving classification speed without detection accuracy degradation. The proposed architecture adopted a hardware and software cooperative design, which is distinctive from other existing architectures. The hardware implementation based on the single-node architecture exploits its resources by using the proposed task scheduling method. Since the task schedules are static within BDT, once one fixed the task schedules of BDT, there is no processing overhead in the detection phase. Also, the task schedule focusing on the lower bound of required cycles clarified that the efficient parallel degree is less than the number of feature channels used in ACF, and made it possible to reduce the hardware resource for leaf nodes without lowering processing performance.

Chapter 6

Hardware Accelerator for Aggregated Channel Features

So far, this dissertation has explained algorithms and hardware architectures for ACF hardware implementation in Chapters 3, 4, and 5. Based on these, this chapter constructs a hardware accelerator for ACF and evaluates its performance on FPGA [80].

6.1 Introduction

In object detection hardware implementation, it has been difficult to satisfy the requirements of detection accuracy and detection speed simultaneously. As mentioned in Chapter 1, this work uses ACF as a baseline to exploit its reasonably high accuracy and low computational cost. For enhancing throughput and minimizing hardware cost, this work has performed algorithm-hardware co-optimization and improved the compatibility of ACF with the hardware implementation explained in Chapters 3, 4, and

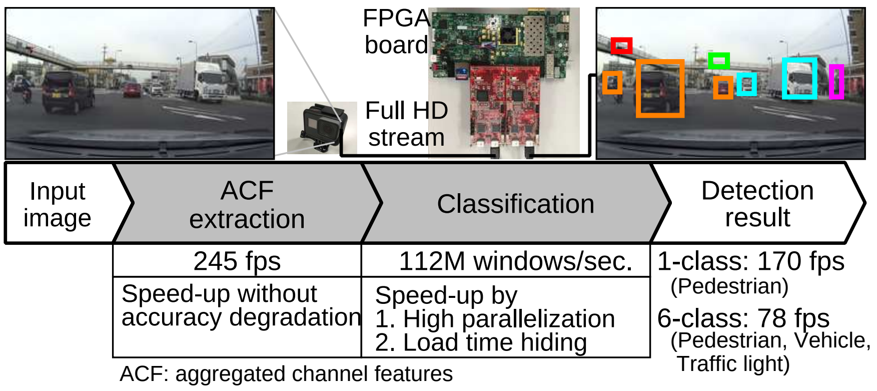


Figure 6.1: Proposed object detection system overview.

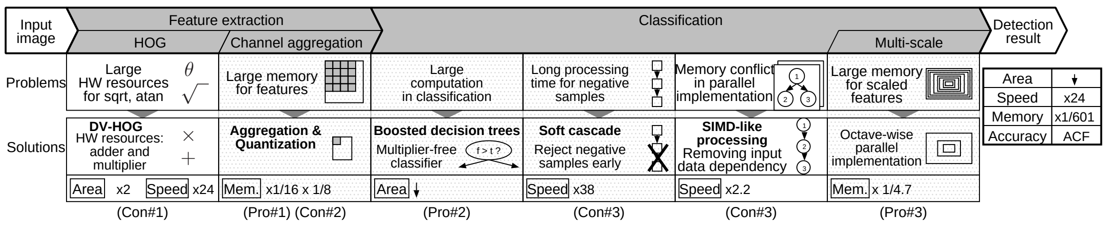

Input image
Feature extraction
HOG
Channel aggregation
Classification
Multi-scale
Detection result
Problems
Large HW resources for sqrt, atan
Large memory for features
Large computation in classification
Long processing time for negative samples
Memory conflict in parallel implementation
Large memory for scaled features
Solutions
DV-HOG HW resources: adder and multiplier
Aggregation & Quantization
Boosted decision trees
Multiplier-free classifier
f > t ?
Soft cascade
Reject negative samples early
SIMD-like processing
Removing input data dependency
Octave-wise parallel implementation
Area x2 Speed x24
Mem. x1/16 x 1/8
Area
Speed x38
Speed x2.2
Mem. x 1/4.7
(Con#1)
(Pro#1) (Con#2)
(Pro#2)
(Con#3)
(Con#3)
(Pro#3)
Area
Speed x24
Memory x1/601
Accuracy ACF

Figure 6.2: Problems and solutions for ACF hardware implementation.

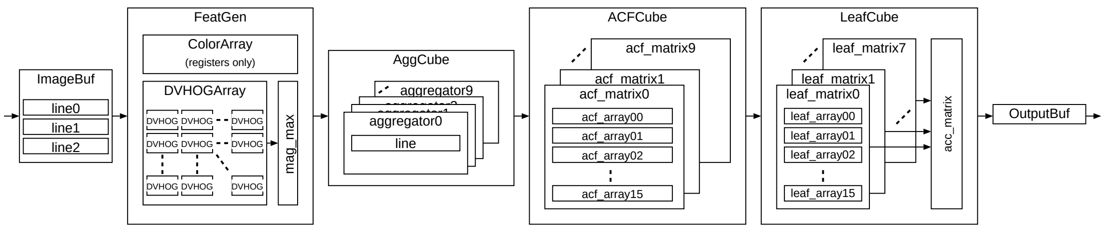


Figure 6.3: ACF-Core hardware architecture.

5. Based on those proposed in the previous chapters, this chapter proposes a general object detection system shown in Figure 6.1 and presents its FPGA implementation. Feature extraction of ACF is speeded up by adopting a hardware-oriented feature descriptor which extracts equivalent information in a small amount of computation as explained in Chapter 3. BDT classification is speeded by parallel implementation and hiding load time of coefficients, which was proposed in Chapter 5, and 112M windows/sec. is attained. In addition, a quantization method which is robust to accuracy degradation explained in Chapter 4 is adopted for memory saving and power reduction. Consequently, the proposed system can detect multi-objects of pedestrians, vehicles, and traffic signals in 1080p60, which satisfies the requirement for the automatic braking system.

The rest of this chapter is organized as follows. Section 6.2 explains the proposed hardware accelerator for ACF. Sections 6.3 and 6.4 evaluate the proposed hardware accelerator, and Section 6.5 concludes this chapter.

6.2 Proposed Hardware Accelerator Architecture

This section presents the proposed architecture for ACF-based object detection. Figure 6.2 summarizes the problems of object detection accelerator and their solutions. Each solution contributes to at least either of area reduction, speed-up, or memory reduction. The advantages of ACF (Pro#1) to (Pro#3) provide memory reduction in channel aggregation, area reduction and fast classification in classification, respectively. (Con#1) to (Con#3), on the other hand, are resolved by the proposed architecture. As

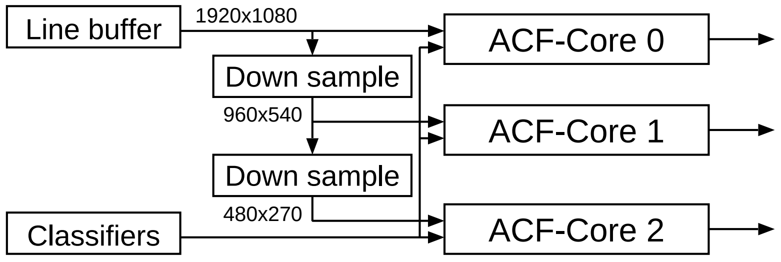


Figure 6.4: Multi-scale ACF hardware architecture.

a result, the proposed accelerator achieves area reduction, 24- and 83.6-times speed-up in feature extraction and classification, respectively, and memory reduction to 1/601 while keeping the detection accuracy almost identical to the original software ACF implementation.

Figure 6.3 shows an ACF object detection core module named `ACF-Core`. It mainly consists of five modules and a control module. First, the `ImageBuf` module stores the input image. The `FeatGen` module extracts DV-HOG, magnitude, and three color channels. DV-HOG feature extraction is parallelized in 3-D: 2-D for image and 1-D for color channels. Then, `AggCube` module aggregates the features extracted in the previous module. `AggCube` has ten channels of memory. Then, `ACFCube` and `LeafCube` modules explained in Chapter 5 process BDT classification. Finally, the positive detection result is output via `OutputBuf` module. In the case of multi-class detection, all the classes share the same aggregated channel features, whereas the classifiers are different. ACF computation for all the classes is performed followed by classification for each class, where the hardware implementation benefits in reducing hardware resources and the computations for feature extraction thanks to sharing features between all the classes.

Thanks to the hardware solutions explained so far, the feature extraction and classification are implemented in parallel for fast detection. For detecting multiple sizes of objects, on the other hand, multi-scale detection, which scales the input image or applies classifiers of different window sizes, must be implemented. In this case, the processing time for the feature extraction would become the bottleneck compared with that of classification. To reduce the feature extraction time, this architecture adopts an approach proposed by Benenson *et al.* [37]. Figure 6.4 shows the overview, where the scale octave is 1/2 and classifiers dedicated for each scaled image are applied to ACF-Core 0 to 2 in parallel. Although a Full HD image has four octaves, the smallest octave image is omitted because it is rare for ADAS to appear large objects. For the convenience of the explanation, this module is named `ACF-HW` module.

6.3 Evaluation using Logic Simulator

This section evaluates the hardware architecture of ACF explained so far.

Table 6.1: Processing performance comparison with conventional implementations.

Method		ACF [36]	ACF [40]	DPM [51]	This work
Platform		CPU	FPGA	ASIC	FPGA
Image size	$W_{\text{img}} \times H_{\text{img}}$	640×480	640×480	$1{,}920 \times 1{,}080$	$1{,}920 \times 1{,}080$
Window size	$W_{\text{win}} \times H_{\text{win}}$	48×96	32×64	64×128	48×96
Scale step	S_{scale}	$2^{1/8}$	$2^{1/6}$	$2^{1/3}$	$2^{1/8}$
Pixel step	S_{pixel}	4	4	8	4
fps	N_{fps}	31.9	30	60	350
#window/sec.	N_{wps}	2,181k (1/105.0)	1,972k (1/116.1)	3,975k (1/57.6)	229,079k (1.0)

6.3.1 Processing Performance

For the evaluation of processing performance, the RTL implementation is simulated with actual input images on ModelSim SE-64 10.3. In the simulation, the classification process takes 45,809 cycles or 0.46 milliseconds for a Full HD image without scaling. Since processing time is linearly proportional to the image resolution, when the processing time t_{single} represents the required cycle or time for a single-scale Full HD image, the entire processing time t_{all} for full search detection with sliding-window sampling is defined as follows:

$$t_{\text{all}} = \sum_{i=1}^{N_{\text{scale}}} \frac{t_{\text{single}}}{S_{\text{scale}}^{2i}}, \tag{6.1}$$

where N_{scale} is the number of scale images. Given t_{single} of 0.46 and the parameters shown in Table 6.1, the processing time t_{all} becomes 2.86 milliseconds. Thus, the proposed hardware enables to process Full HD images at 350 fps.

Table 6.1 provides a processing performance comparison between the proposed method and three conventional methods: an ACF software implementation [36], an ACF hardware implementation [40], and a deformable part model (DPM) hardware implementation [51]. The DPM hardware implementation is the fastest hardware implementation so far, using a deformable part model [51]. Evaluation based on frame rate does not provide precise result because it does not use detailed implementation settings [81]. Therefore, window-based evaluation is suitable for a fair comparison, and it is adopted here. The processing performance is evaluated by recalculating to processed windows per second, N_{wps}, using the following equation:

$$N_{\text{wps}} = \frac{N_{\text{fps}}}{S_{\text{pixel}}^{2}} \sum_{i=1}^{N_{\text{scale}}} \left(\frac{W_{\text{img}}}{S_{\text{scale}}^{i}} - W_{\text{win}} \right) \left(\frac{H_{\text{img}}}{S_{\text{scale}}^{i}} - H_{\text{win}} \right). \tag{6.2}$$

As shown in Table 6.1, the proposed implementation is 105.0 and 116.1 times faster than the software and the hardware implementations of ACF, respectively. Also, it is

Table 6.2: Resource usage of feature extraction with 32 degrees of parallelism.

Feature	Slice LUTs	Slice registers	Throughput
HOG [8]	12,288 (5.6%)	1,440 (0.3%)	32 features/cycle
RGB	6,144 (2.8%)	6,144 (1.4%)	32 features/cycle
Magnitude	2,272 (1.0%)	1,764 (0.4%)	32 features/cycle

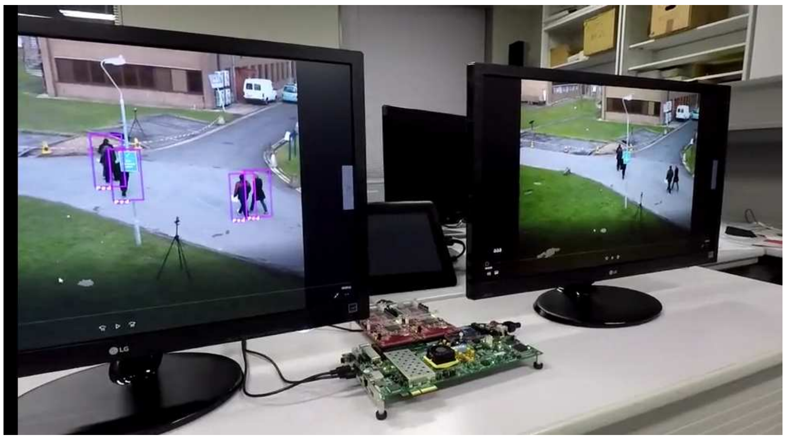


Figure 6.5: FPGA implementation environment.

57.6 times faster compared with the DPM hardware implementation, which was the fastest implementation.

6.3.2 Discussion

The processing performance of practical applications depends on both feature extraction and classification. So far, this section has shown the proposed BDT hardware architecture can process 350 fps for Full HD images. Now, let us discuss the processing performance of the feature extraction part. Suppose three feature descriptors mentioned above. Table 6.2 summarizes the implementation result with 32 degrees of parallelism, where the parallelism comes from eight channels and four scaled images from a Full HD image. As shown in Table 6.2, the entire utilization of feature extraction modules is less than 10% of slices. With 32 degrees of parallelism, the feature extraction modules can achieve 60 fps processing performance for Full HD images. However, it does not seem to be enough for providing feature maps to the proposed BDT classifier that can achieve 350 fps. Besides, to realize *N*-class object detection, the proposed BDT classifier needs to process *N* times faster than the feature extraction modules. In this case, the proposed BDT classifier with the assumed feature extraction modules can classify five classes of objects simultaneously.

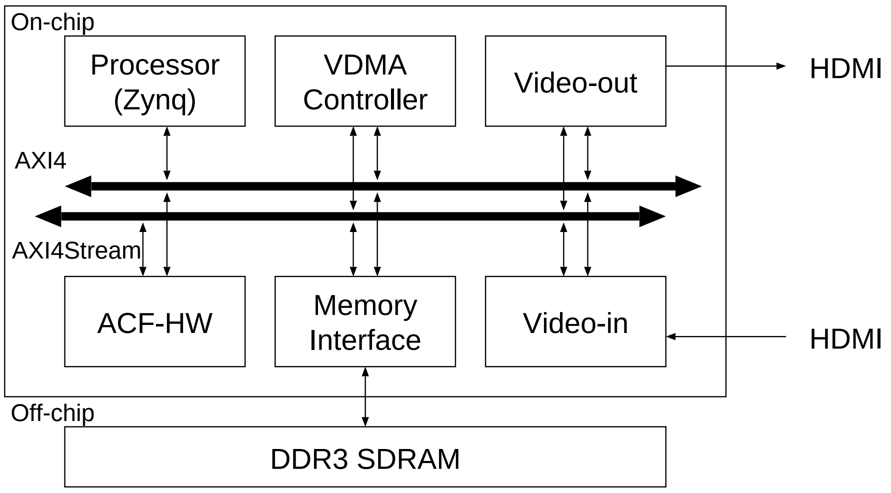


Figure 6.6: FPGA implementation overview.

Table 6.3: Parameters used in the evaluation.

Module		Parameter	Value
Feature extraction		Parallelism	24
		Bit-width	4bit
Classification	ACF-Core0	Parallelism	8x16x8
	ACF-Core1		8x4x8
	ACF-Core2		8x2x8

6.4 FPGA Implementation

The proposed hardware architecture is implemented on FPGA. The evaluation uses a Xilinx ZC706 evaluation board and two FMC cards for HDMI input and output as shown in Figure 6.5. The FPGA board has programmable logic (PL) and ARM core. Figure 6.6 shows an overview of FPGA implementation. The ACF-HW module is implemented in register transfer level using Verilog HDL in PL, and it is wrapped as a custom IP which is controllable via AXI interfaces. The FMC cards used in the evaluation supports 1080p60 at most, and the object detection system is designed to support object detection for this speed. To handle the 1080p60 input stream and overlay the bounding boxes on the output stream, the implementation used the Vivado IPs of a video input, a video output, and a video timing controller. The trained classifier is stored in an SD card, and software running on the ARM core loads it.

Table 6.3 shows the parameters used in the implementation, and the target frequency for the ACF-HW module is 100MHz. Table 6.4 summarizes the resource utilization, which shows the balanced usage of resources. It can be seen that DV-HOG modules occupy 8% of LUTs in total. It should be noted, on the other hand, that the original

Table 6.4: FPGA resource utilization.

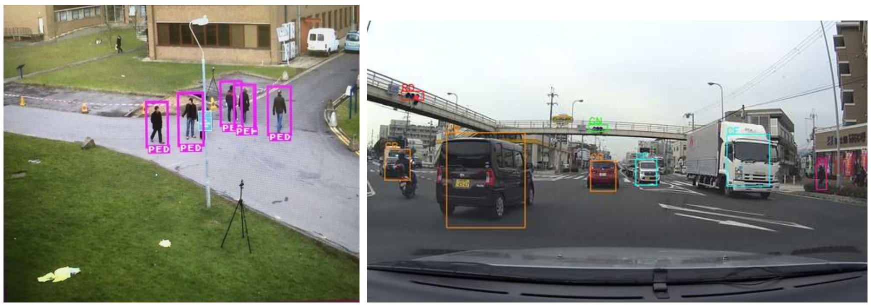
Module	Slice		LUT (Logic)		LUT (Memory)	
ACF-HW	124,770	(57%)	124,476	(57%)	294	(0%)
→ ACF-Core0	70,755	(32%)	70,657	(32%)	98	(0%)
→ ImageBuf	195	(0%)	195	(0%)	0	(0%)
→ FeatGen	6,928	(3%)	6,831	(3%)	97	(0%)
→ DV-HOG	6,168	(3%)	6,168	(3%)	0	(0%)
→ AggCube	2,021	(1%)	2,021	(1%)	0	(0%)
→ ACFCube	23,891	(11%)	23,891	(11%)	0	(0%)
→ LeafCube	35,144	(16%)	35,143	(16%)	1	(0%)
→ OutputBuf	39	(0%)	39	(0%)	0	(0%)
→ ACF-Core1	26,127	(12%)	26,119	(12%)	98	(0%)
→ ACF-Core2	18,859	(9%)	18,761	(9%)	98	(0%)
Total	139,215	(64%)	137,939	(63%)	1,276	(2%)

Module	LUT (FF)		32Kb BRAM		DSP	
ACF-HW	128,626	(59%)	356.5	(65%)	122	(14%)
→ ACF-Core0	72,719	(33%)	204.5	(38%)	41	(5%)
→ ImageBuf	601	(0%)	9	(2%)	0	(0%)
→ FeatGen	8,215	(4%)	0	(0%)	0	(0%)
→ DV-HOG	7,457	(3%)	0	(0%)	0	(0%)
→ AggCube	2,878	(1%)	8.5	(2%)	40	(4%)
→ ACFCube	24,420	(11%)	170	(31%)	0	(0%)
→ LeafCube	36,839	(17%)	16	(3%)	0	(0%)
→ OutputBuf	1,086	(0%)	1	(0%)	0	(0%)
→ ACF-Core1	27,158	(12%)	84.5	(16%)	41	(5%)
→ ACF-Core2	19,686	(9%)	64.5	(12%)	40	(4%)
Total	149,129	(68%)	389	(71%)	128	(14%)

computation requires 14x resources, and the corresponding resource reaches 112%, which makes a single FPGA implementation infeasible. Taking into account that 64% of BRAM are used for ACF memory in total, it is impossible to store the aggregated channel features without the quantization method, which requires 512% of memory. In terms of BDT classification, the 1,024 parallel classification contributes to 845-times speed up with only 27% of slices usage, which significantly improved the trade-off between hardware resources and detection speed of BDT.

6.4.1 Performance Evaluation on Pedestrian Detection

The evaluation uses twelve classifiers whose window size ranges from 48x96 to 92x184, and each classifier consists of 2,048 depth-two decision trees. Training



(a) Pedestrian detection. (b) Traffic object detection.

Figure 6.7: Detection results.

Table 6.5: Detection performance comparison.

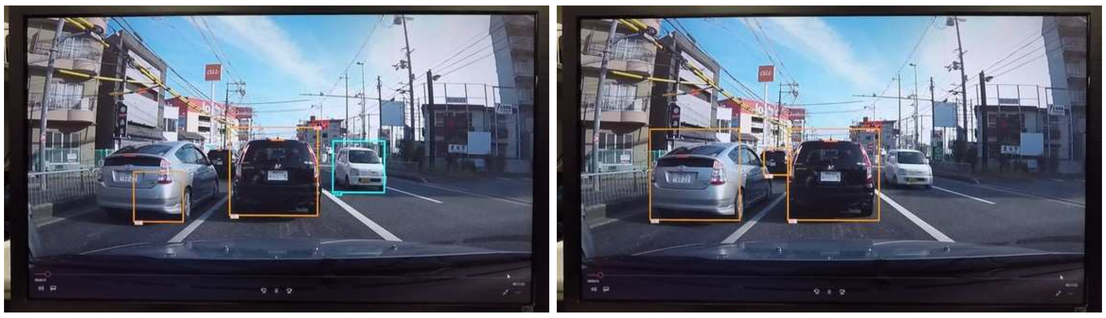
Method	Speed	Method	log-avg. MR	#win. / sec.
Suleiman *et al.* [13]	Full HD 60 fps	SVM	46%	6,284k
Suleiman *et al.* [51]	Full HD 60 fps	DPM	20%	3,975k
Song *et al.* [40]	VGA 30 fps	ACF	17%	1,972k
This work	Full HD 170 fps	ACF	17%	112,501k

process uses INRIA Person Dataset [24]. Figure 6.7(a) shows the detection result. For quantitative analysis, software simulation is used to count clock cycles for each step. The result shows that feature extraction and classification consume 4.08 and 1.80 (= 0.15×12(scales)) milliseconds, respectively. Consequently, the proposed accelerator can process 170 fps of Full HD. In the implemented system, the bottleneck of the processing performance is the input part that receives images of the 1080p60 video stream from HDMI. The speed of the input stream is fixed, and it restricts the maximum parallelism of feature extraction.

Table 6.5 shows the processing performance comparison. [13] and [51] achieve Full HD 60 fps processing. However, they suffer from the higher log-average MR of 46% and 20% on INRIA Person Dataset, respectively, whereas the accuracy of the proposed architecture is 17%. For a fair comparison in terms of the processing speed of these detection systems, the evaluation uses the number of processing windows in a second

Table 6.6: Classifiers for traffic objects.

Target	Pedestrian	Vehicle (Front, Rear)	Traffic light (Green, Yellow, Red)
Depth	2	2	2
#weak classifier	2,048	512	512
Window size	[48, 96], ..., [92,184]	[48, 48], ..., [92,92]	[48, 16], ..., [84,28]
#classifier	12	12	4
Total #classifier	12	24	12
Area	Lower 2/3	Lower 2/3	Upper half



(a) Detection result. (b) Detection result of the (a)'s next frame.

Figure 6.8: Unstable detection results of traffic objects.

Table 6.7: Classification speed evaluation.

		# cycle		Speed-up
		Soft cascade		
		Off	On	
Vehicle	Front	148,677	13,626	10.9x
	Rear	153,143	15,193	10.1x
Pedestrian		575,037	14,996	38.3x
Traffic Light	Green	156,133	9,481	16.5x
	Yellow	116,308	10,716	10.9x
	Red	140,734	9,403	15.0x
Total		1,290,032	73,415	17.6x

as an evaluation metric. This is because the evaluation based on frame rates does not reflect parameters such as the number of images in an octave and the detection window stride, which largely influences the detection performance. For accurate evaluation, the evaluation metric is calculated by using the parameters reported in each paper. Table 6.5 indicates that the proposed accelerator achieves 57 times speed-up compared with the existing ACF hardware [40].

6.4.2 Traffic Object Detection

This section aims to analyze the processing performance of the proposed detection system to the traffic objects in the urban areas of Japan supposing the practical ADAS. Multi-class detection for pedestrian, vehicle, and traffic light, which are fundamental traffic objects, are performed on FPGA. To the best of my knowledge, there exists no public traffic object dataset recorded in Japan. Although traffic datasets taken outside Japan such as KITTI [63], Cityscapes [82], and urban object detection dataset [83] are available, it is not suitable to apply them to the evaluation because the appearances of traffic objects differ in countries. Thus, for training and testing traffic lights and vehicles, this evaluation uses a private dataset recorded in Japan. As for the pedestrian detection, the evaluation uses the same classifier explained in Section 6.4.1, which is trained by INRIA Person Dataset. Aiming to the quantitative evaluation, this research is constructing a Japanese traffic object dataset by annotating the three traffic objects from images taken in the Japanese traffic environment. Table 6.6 summarizes the classifiers used in the evaluation. The detection candidate area is limited as shown in Table 6.6 to reduce the number of false positives and speed-up.

Figure 6.7(b) shows the detection result, which was obtained without object tracking. Table 6.7 summarizes the processing time. It can be seen that soft cascade contributes to 17.6 times speed-up on average. The system can process 78 fps of Full HD frames, which is enough for the driving assistance system. It should be noted that the proposed accelerator does not use any domain-specific knowledge and hence it is applicable to any object detection applications, whereas the required frame rate for the driving assistance system is exemplified above. Figure 6.8 compares the detection results of consecutive frames. Although most objects are correctly detected on average, missed objects and incorrect results in terms of positions and sizes are observed in some frames as shown in the right vehicle of Figure 6.8(b) and the left vehicle of Figure 6.8(a), respectively. These unstable results originate from a slight movement of the object and luminance change. These detection misses can be overcome by using the object tracking that interpolates detection results between consecutive frames. This will be discussed in Chapter 7 as future work.

6.5 Summary

Embedded object detection systems need to simultaneously achieve high detection accuracy, fast detection, and low power consumption, and its design is highly challenging. To solve the issue, this chapter proposed a hardware architecture for general multi-class object detection using ACF. The proposed hardware architecture makes use of the advantages of the ACF algorithm itself and incorporates multiplier-free DV-HOG, aggressive quantization and BDT parallel computation architecture with the overall accelerator architecture. In total, the system is speed-up by 24 and 83 times for feature extraction and classification, and reduced memory to 1/600. FPGA implementation result showed that the proposed system could detect pedestrians in 170 fps for a Full HD image, and 6-class traffic objects in 78 fps for Full HD, which satisfied the requirement for the automatic braking system.

Chapter 7

Conclusion

7.1 Summary and Conclusions

In object detection hardware, achieving both high detection accuracy and fast detection is a challenging issue. This dissertation has proposed a hardware architecture for object detection, which improves the trade-off of speed and detection accuracy. This dissertation has focused on the information-preservation in architecture design for ensuring no accuracy degradation and efficient hardware resource utilization for speed-up. The FPGA implementation result shows that for a Full HD input video stream, the system can process 170 fps for pedestrian detection, and 78 fps for 6-class traffic object detection.

Chapter 3 discusses an information-preserved HOG algorithm and its hardware architecture. The proposed DV-HOG algorithm is based on linear algebra and requires only additions and multiplications with constant. The analysis and the evaluation result validate the information-preservation. The proposed hardware architecture utilizes the symmetry of x and y-axes to reduce hardware resources. The implementation result shows that the required hardware resources are reduced to 1/12 without accuracy degradation. The proposed DV-HOG improved the trade-off between detection accuracy and power consumption.

Chapter 4 provides an analysis of the influence of numerical precision on classification accuracy. The preliminary evaluation result shows that AdaBoost is robust to limited numerical precision because of its narrow region of interest compared with MLP and SVM. Based on this analysis, Chapter 4 presents an aggressive quantization method for BDT. The quantization method focuses only on the range of the threshold values. The evaluation result shows that aggressive quantization from 32-bit to 2-bit degrades only 2% log-average MR on INRIA Person Dataset. The proposed method contributes to reducing the memory requirement of classifiers on embedded systems.

Chapter 5 describes a hardware architecture for parallel BDT classification. The proposed hardware architecture utilizes the three-dimensional parallelism: 2-D for image resolution, and 1-D for feature channels. For high dimensional parallel classification, this chapter also proposes a task scheduling algorithm to avoid memory access conflict. The task scheduling algorithm determines the memory access order by the clock cycle level. The 1,024-parallelly implemented hardware architecture achieves 6.6 times speed-up. The evaluation result also shows that the task scheduling algorithm is effective for negative samples because it is compatible with soft cascade.

Chapter 6 explains a hardware accelerator for ACF. The system uses the algorithms and hardware architectures as key components proposed in Chapters 3, 4, and 5, which resolves the difficulties of implementing ACF hardware. The synthesis result on FPGA shows the balanced resource utilization, and the proposed system enables to operate object detection in high parallelism. The implemented system achieves both reasonably high detection accuracy and fast speed for multi-class classification.

This dissertation proposed the DV-HOG feature descriptor and a quantization method for a BDT classifier based on information preserved algorithms, and a BDT hardware architecture for efficient object detection hardware implementations. The information preserved algorithm focuses only on the data required for classification and uses only hardware orientations computations. This information preservation approach makes it possible to improve the hardware implementation efficiency. The BDT hardware architecture and the task scheduling approach enhanced the hardware resource utilization and contribute to fast classification.

7.2 Future Works

The implemented system achieves high detection speed enough for practical applications with limited hardware resources. However, there still remains room for improving the detection accuracy. For example, the single shot detector (SSD) [18], which is one of the state-of-the-art detection method based on deep neural network, achieves less than 10% log-average MR on INRIA Person Dataset. The difference in log-average MR between SSD and ACF is more than 5%. This section discusses two promising approaches to improve detection accuracy: one is to improve detection accuracy of a classifier itself, and the other is to complement detection accuracy by combining with object tracking.

Many novel object detection algorithms based on deep neural networks achieve state-of-the-art detection accuracy. The representation capability of deep learning is derived from deeply accumulated network structures. Some recent researches propose deep decision tree classifiers for both making use of representation capability and reducing computations [43, 84, 85]. Deep forest [43] consists of multiple layers of decision tree based classifier, and it achieves the equivalent detection accuracy to deep neural networks. The proposed hardware architecture is compatible with any binary decision trees. Thus, updating classifiers or modifying the BDT structure for increasing representation capability is one promising approach for detection accuracy improvement. Especially for time-critical embedded systems, the small amount of computations of decision trees is a significant benefit.

Use of time continuity for object detection is also a promising approach to detection accuracy improvement. Object tracking aims to find the correspondence of objects in consecutive frames of an input video. Object tracking is able to trace objects correctly even when detection fails such as occlusion and changing light condition. Combining object tracking and object detection is called *detection-by-tracking* and *tracking-by-detection* and many researchers have proposed such methods [86–90]. [90] reports a BDT-based object tracking method, which uses the similarity of BDT's selected path as a feature for object tracking. This tracking method uses only the index of the selected leaf node, and it is highly compatible with the proposed hardware architecture. In applying this method to the proposed hardware, more memory resources are required for storing the selected index of BDT. This increase has a significant effect, especially for on multi-class classification. Future work includes updating the object tracking method using BDT for memory reduction and hardware implementation.

Bibliography

[1] Z. Sun, G. Bebis, and R. Miller, “On-road vehicle detection: a review,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 5, pp. 694–711, May 2006.

[2] T. Gandhi and M. M. Trivedi, “Pedestrian protection systems: Issues, survey, and challenges,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 3, pp. 413–430, Sep. 2007.

[3] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, “Survey of pedestrian detection for advanced driver assistance systems,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 7, pp. 1239–1258, Jul. 2010.

[4] S. Sivaraman and M. M. Trivedi, “Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 4, pp. 1773–1795, Dec. 2013.

[5] 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,” in *Proceedings of IEEE Workshop on Signal Processing Systems*, Oct. 2012, pp. 197–202.

[6] K. Mizuno, K. Takagi, Y. Terachi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, “A sub-100 mW dual-core HOG accelerator VLSI for parallel feature extraction processing for HDTV resolution video,” *IEICE Transactions on electronics*, vol. 96, no. 4, pp. 433–443, Apr. 2013.

[7] M. Hiromoto and R. Miyamoto, “Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features,” in *Proceedings of IEEE International Conference on Computer Vision Workshops*, Sep. 2009, pp. 894–899.

[8] P.-Y. Chen, C.-C. Huang, C.-Y. Lien, and Y.-H. Tsai, “An efficient hardware implementation of HOG feature extraction for human detection,” *IEEE*

Transactions on Intelligent Transportation Systems, vol. 15, no. 2, pp. 656–662, Apr. 2014.

[9] L. Maggiani, C. Bourrasset, F. Berry, J. Sérot, M. Petracca, and C. Salvadori, "Parallel image gradient extraction core for FPGA-based smart cameras," in *Proceedings of International Conference on Distributed Smart Cameras*, Sep. 2015, pp. 128–133.

[10] L. Maggiani, C. Bourrasset, M. Petracca, F. Berry, P. Pagano, and C. Salvadori, "HOG-Dot: A parallel kernel-based gradient extraction for embedded image processing," *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 2132–2136, Nov. 2015.

[11] S. Lee, H. Son, J. C. Choi, and K. Min, "HOG feature extractor circuit for real-time human and vehicle detection," in *Proceedings of IEEE Region 10 Conference*, Nov. 2012, pp. 1–5.

[12] M. Hemmati, M. Biglari-Abhari, S. Berber, and S. Niar, "HOG feature extractor hardware accelerator for real-time pedestrian detection," in *Proceedings of Euromicro Conference on Digital System Design*, Aug. 2014, pp. 543–550.

[13] A. Suleiman and V. Sze, "An energy-efficient hardware implementation of HOG-based object detection at 1080HD 60 fps with multi-scale support," *Journal of Signal Processing Systems*, vol. 84, no. 3, pp. 325–337, Sep. 2016.

[14] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006.

[15] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," in *Proceedings of International Conference on Artificial Intelligence and Statistics*, Apr. 2009, pp. 448–455.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proceedings of Advances in Neural Information Processing Systems*, Dec. 2015, pp. 91–99.

[17] L. Zhang, L. Lin, X. Liang, and K. He, "Is faster R-CNN doing well for pedestrian detection?" in *Proceedings of European Conference on Computer Vision*, Oct. 2016, pp. 443–457.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proceedings of European Conference on Computer Vision*, Oct. 2016, pp. 21–37.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp. 779–788.

[20] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jul. 2017, pp. 936–944.

[21] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jul. 2017, pp. 6517–6525.

[22] G. Brazil, X. Yin, and X. Liu, "Illuminating pedestrians via simultaneous detection and segmentation," in *Proceedings of IEEE International Conference on Computer Vision*, Oct. 2017, pp. 4960–4969.

[23] R. Zhao, X. Niu, Y. Wu, W. Luk, and Q. Liu, "Optimizing CNN-based object detection algorithms on embedded FPGA platforms," in *Applied Reconfigurable Comput.*, Apr. 2017, pp. 255–267.

[24] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2005, pp. 886–893.

[25] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, Apr. 2012.

[26] ——, "Pedestrian detection: A benchmark," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 304–311.

[27] R. Benenson, M. Omran, J. Hosang, and B. Schiele, "Ten years of pedestrian detection, what have we learned?" in *Proceedings of European Conference on Computer Vision Workshop*, Sep. 2015, pp. 613–627.

[28] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *arXiv:1809.02165*, Sep. 2018.

[29] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[30] 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*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[31] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, Jul. 2002.

[32] T. Watanabe, S. Ito, and K. Yokoi, "Co-occurrence histograms of oriented gradients for pedestrian detection," in *Proceedings of Advances in Image and Video Technology*, Jan. 2009, pp. 37–47.

[33] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of IEEE International Conference on Computer Vision*, vol. 2, Sep. 1999, pp. 1150–1157.

[34] W. Nam, P. Dollár, and J. H. Han, "Local decorrelation for improved pedestrian detection," in *Proceedings of Advances in Neural Information Processing Systems*, Dec. 2014, pp. 424–432.

[35] S. Zhang, R. Benenson, and B. Schiele, "Filtered channel features for pedestrian detection," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2015, pp. 1751–1760.

[36] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1532–1545, Aug. 2014.

[37] R. Benenson, M. Mathias, R. Timofte, and L. V. Gool, "Pedestrian detection at 100 frames per second," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 2903–2910.

[38] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255.

[39] S. Zhang, R. Benenson, and B. Schiele, "CityPersons: A diverse dataset for pedestrian detection," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jul. 2017, pp. 4457–4465.

[40] H. Song, B. Jeong, H. Choi, T. Cho, and H. Chung, "Hardware implementation of aggregated channel features for ADAS," in *Proceedings of International SoC Design Conference*, Oct. 2016, pp. 167–168.

[41] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proceedings of Advances in Neural Information Processing Systems*, Dec. 2015, pp. 3123–3131.

[42] F. Li and B. Liu, "Ternary weight networks," *arXiv: 1605.04711*, May 2016.

[43] Z.-H. Zhou and J. Feng, “Deep forest: Towards an alternative to deep neural networks,” in *Proceedings of International Joint Conference on Artificial Intelligence*, Aug. 2017, pp. 3553–3559.

[44] L. Bourdev and J. Brandt, “Robust object detection via soft cascade,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, Jun. 2005, pp. 236–243.

[45] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, “Hardware architecture for HOG feature extraction,” in *Proceedings of International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Sep. 2009, pp. 1330–1333.

[46] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, “FPGA-based real-time pedestrian detection on high-resolution images,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop*, Jun. 2013, pp. 629–635.

[47] S. Bauer, S. Köhler, K. Doll, and U. Brunsmann, “FPGA-GPU architecture for kernel SVM pedestrian detection,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop*, Jun. 2010, pp. 61–68.

[48] J.-F. Wang, C.-S. Choy, T.-L. Chao, K.-C. Kit, K.-P. Pun, W.-L. Ouyang, and X.-G. Wang, “Simplifying HOG arithmetic for speedy hardware realization,” in *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems*, Nov. 2014, pp. 61–64.

[49] S. Bauer and S. S.-M. U. Brunsmann, “FPGA implementation of a HOG-based pedestrian recognition system,” in *Proceedings of MPC Workshop*, Jul. 2009, pp. 49–58.

[50] G.-M. Lozito, A. Laudani, F. Riganti-Fulginei, and A. Salvini, “FPGA implementations of feed forward neural network by using floating point hardware accelerators,” *Advances in Electrical and Electronic Engineering*, vol. 12, no. 1, pp. 30–39, Mar. 2014.

[51] A. Suleiman, Z. Zhang, and V. Sze, “A 58.6 mW 30 frames/s real-time programmable multiobject detection accelerator with deformable parts models on full HD 1920×1080 videos,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 3, pp. 844–855, Mar. 2017.

[52] S. I. Venieris, A. Kouris, and C.-S. Bouganis, “Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions,” *ACM Computing Surveys*, vol. 51, no. 3, pp. 56:1–56:39, Jun. 2018.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Parallel distributed processing: explorations in the microstructure of cognition.* The MIT Press, Jul. 1987, ch. Learning Internal Representations by Error Propagation.

[54] D. D. Gajski, *Principles of digital design.* Prentice Hall New York, 1997.

[55] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[56] K. Piromsopa, C. Aporntewan, and P. Chongsatitvatana, "An FPGA implementation of a fixed-point square root operation," in *Proceedings of International Symposium on Communications and Information Technologies*, 2001, pp. 587–689.

[57] Y. Bengio, O. Delalleau, and C. Simard, "Decision trees do not generalize to new variations," *Computational Intelligence*, vol. 26, no. 4, pp. 449–467, Nov. 2010.

[58] A. Bermak and D. Martinez, "A compact 3D VLSI classifier using bagging threshold network ensembles," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1097–1109, Sep. 2003.

[59] M. Owaida, H. Zhang, C. Zhang, and G. Alonso, "Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms," in *Proceedings of International Conference on Field-Programmable Logic and Applications*, Sep. 2017, pp. 1–8.

[60] R. J. R. Struharik and L. A. Novak, "Hardware implementation of decision tree ensembles," *Journal of Circuits, Systems and Computers*, vol. 22, no. 05, p. 1350032, Jun. 2013.

[61] K. Mitsunari, Y. Takeuchi, M. Imai, and J. Yu, "Decomposed vector histograms of oriented gradients for efficient hardware implementation," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101A, no. 11, pp. 1766–1775, Nov. 2018.

[62] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," in *Proceedings of British Machine Vision Conference*, Sep. 2009, pp. 91.1–91.11.

[63] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 3354–3361.

[64] P. Dollár, S. Belongie, and P. Perona, "The fastest pedestrian detector in the west," in *Proceedings of British Machine Vision Conference*, vol. 2, no. 3, Sep. 2010, pp. 68.1–68.11.

[65] K. Mitsunari and J. Yu, “Influence of numerical precision on machine learning and embedded systems,” in *Proceedings of International Workshop on Smart Info-Media Systems in Asia*, Sep. 2016, pp. 164–169.

[66] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[67] M. Courbariaux, Y. Bengio, and J.-P. David, “Training deep neural networks with low precision multiplications,” in *Proceedings of International Conference on Learning Representations Workshop*, May 2015.

[68] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *Proceedings of International Conference on Machine Learning*, Jul. 2015, pp. 1737–1746.

[69] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, E. McQuinn, B. Shaw, N. Pass, and D. S. Modha, “Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores,” in *Proceedings of IEEE International Joint Conference on Neural Networks*, Aug. 2013, pp. 1–10.

[70] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, “SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor,” in *Proceedings of IEEE International Joint Conference on Neural Networks*, vol. 5, Jun. 2008, pp. 2849–2856.

[71] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: the RPROP algorithm,” in *Proceedings of IEEE International Conference on Neural Networks*, vol. 1, Mar. 1993, pp. 586–591.

[72] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: a statistical view of boosting,” *The Annals of Statistics*, vol. 28, no. 2, pp. 337–407, Apr. 2000.

[73] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, “An improved GLMNET for L1-regularized logistic regression,” *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1999–2030, Jun. 2012.

[74] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror, “Result analysis of the NIPS 2003 feature selection challenge,” in *Proceedings of Advances in Neural Information Processing Systems*, Dec. 2005, pp. 545–552.

[75] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std. 754-2008, Aug. 2008.

[76] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, May 2011.

[77] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[78] K. Mitsunari, J. Yu, T. Onoye, and M. Hashimoto, “Hardware architecture for high-speed object detection using decision tree ensemble,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101A, no. 9, pp. 1298–1307, Sep. 2018.

[79] E. Ohn-Bar and M. M. Trivedi, “To boost or not to boost? on the limits of boosted trees for object detection,” in *Proceedings of International Conference on Pattern Recognition*, Dec. 2016, pp. 3350–3355.

[80] K. Mitsunari, J. Yu, and M. Hashimoto, “Hardware architecture for fast general object detection using aggregated channel features,” in *Proceedings of IEEE Asian Solid-State Circuits Conference*, Nov. 2018, pp. 55–58.

[81] X. Ma, W. A. Najjar, and A. K. Roy-Chowdhury, “Evaluation and acceleration of high-throughput fixed-point object detection on FPGAs,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 6, pp. 1051–1062, Jun. 2015.

[82] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The cityscapes dataset for semantic urban scene understanding,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp. 3213–3223.

[83] A. Dominguez-Sanchez, M. Cazorla, and S. Orts-Escolano, “A new dataset and performance evaluation of a region-based CNN for urban object detection,” *Electronics*, vol. 7, no. 11, Nov. 2018.

[84] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulo, “Deep neural decision forests,” in *Proceedings of IEEE International Conference on Computer Vision*, Dec. 2016, pp. 1467–1475.

[85] D. Ignatov and A. Ignatov, “Decision stream: Cultivating deep decision trees,” in *Proceedings of IEEE International Conference on Tools with Artificial Intelligence*, Nov. 2018, pp. 905–912.

[86] M. Andriluka, S. Roth, and B. Schiele, “People-tracking-by-detection and people-detection-by-tracking,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2008, pp. 1–8.

[87] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. V. Gool, "Robust tracking-by-detection using a detector confidence particle filter," in *Proceedings of IEEE International Conference on Computer Vision*, Sep. 2009, pp. 1515–1522.

[88] ——, "Online multiperson tracking-by-detection from a single, uncalibrated camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 9, pp. 1820–1833, Sep. 2011.

[89] E. Moussy, A. A. Mekonnen, G. Marion, and F. Lerasle, "A comparative view on exemplar 'tracking-by-detection' approaches," in *Proceedings of IEEE International Conference on Advanced Video and Signal Based Surveillance*, Aug. 2015, pp. 1–6.

[90] K. Mitsunari, J. Yu, Y. Takeuchi, and M. Imai, "Object tracking based on path similarity of boosted decision trees," in *Proceedings of International Technical Conference on Circuits/Systems, Computers and Communications*, Jul. 2016, pp. 563–566.