Study on Power-Efficient Acceleration Coprocessors in CMOS Technology for Real-time Embedded Machine Learning and Vision

（リアルタイム組込み機械学習及びビジョンのための CMOS 技術における電力効率の良い加速コプロセッサに関する研究）

広島大学大学院工学研究科

システムサイバネティクス専攻

博士課程後期

D161378 張　湘煜

Contents

List of Figures

Abstract

Humans can detect and identify a multitude of objects in a scene with little effort, despite the wide variability in appearance of outdoor scenes, such as complex backgrounds, different poses and illumination conditions. However, it is still challenging for a computer to recognize objects in an image or video sequence. Many efforts have been made to solve the task over multiple decades. Object recognition plays a key role in various fields, including smart vehicle technologies such as advanced driver assistance systems (ADAS), advanced human computer interfaces, robotics, surveillance, security, and intelligent transportation systems. For example, the ADAS uses the images acquired from a camera mounted on the vehicle, detecting the pedestrians and vehicles in images. The recognized results can be utilized to prevent accidents. Pedestrian detection is a challenging work because of various clothes, changing lighting conditions, viewpoints, and a wide range of people's positions and sizes. Recent studies have shown that the sliding-window based methods can overcome these obstacles and obtain improved recognition performances. A feature descriptor extracts the representative data of an image, improving perception of the surrounding environment and transforming the sensed signals into a suitable data format required for the subsequent recognition processing. The histogram of orientation gradients (HOG) algorithm was proposed to distinguish pedestrians in images, obtaining good recognition results through extracting a robust feature set based on gradients. HOG features have been widely applied in image classification and scene understanding tasks.

However, the huge computational complexity of the HOG descriptor remains as a problem for the processing speed. An image has plenty of the sliding windows, and there are several thousand dimensional features in each sliding window. The enormous complexity of computation makes its application almost impossible for real-time processing. Studies have shown that the HOG descriptor can obtain good recognition performance and processing speed in a PC environment. However, the vehicle and robot systems, which use an embedded processor platform, have no access to the same computing resources as PC environments. The feature space is too large for embedded systems and the processing power of an embedded platform is much lower than that of a PC platform, making it hard for systems based on embedded platforms to recognize objects in real-time.

In addition to recognition, learning internal representations of the perceived environment is essential because it bridges the gap between the representations of the object and the data needed by the computer to perform its task. Learning vector quantization (LVQ) neural

networks have been successfully used for a broad range of technical applications, such as image compression and object recognition. In the literature, the LVQ was implemented in software off-line on computer systems or embedded processors. Unfortunately, the software-based approaches cannot deliver reasonable performance for online learning due to the high cost of computational requirements.

Hardware-based accelerators for computer vision can meet the low power and real-time processing requirements for mobile devices because the accelerator architecture can be tailored to specific applications and can be massively parallelized. A high energy efficiency and high throughput are the advantages over software-based solutions and general-purpose hardware.

In this thesis, I present two application specific integrated circuits (ASICs), aiming at low power, portable, and real-time applications and markets, such as Advanced Driver Assistance System (ADAS), robotics, drones, or mobile phones. One is an inference system designed to extract features from images using the HOG feature extractor and to reason based of the results of a nearest-neighbor-search (NNS) classifier. The other is an online learning system aimed to learn from the images, featuring an LVQ neural network. The ability to reason and the ability to learn are the two major capabilities associated with these systems.

The inference system combines the feature extraction and dimension reduction in an intermediate step using partial-least-squares-regression in order to avoid the curse of high dimensionality. The design reduces the redundancy in original feature vectors, converting high-dimensional feature-vectors into low-dimensional feature-vectors. The following NNS works on feature vectors in a reduced-dimension space. The developed hardware-oriented algorithm exploits the cell-based scan strategy which enables image-sensor synchronization and extraction-speed acceleration. Furthermore, buffers for image frames or integral images are avoided. The fabricated test chip in 180 nm CMOS technology achieves fast processing speed and large flexibility for different image resolutions with substantially reduced hardware cost and energy consumption. For the application example of XGA (1024 × 768) resolution videos, HOG-feature vectors can be extracted at 120 MHz operating frequency with a maximum frame rate of 122 fps. An improved version was fabricated in 65 nm CMOS technology which can process XGA (1024 × 768, 30 fps) video in real time, achieving 50 MHz feature extraction and 200 MHz classification, with energy consumption of 906 pJ/pixel. Detection accuracy can be improved using complementary features in addition to the HOG feature, at the cost of an extra 40% power consumption, 64% area requirement, and 53% memory size.

The online learning system is based on a modular and reconfigurable pipeline architecture (MRPA) for LVQ. The MRPA consists of dynamically reconfigurable modules and realizes a

run-time and on-chip configuration for recognition and learning. The developed architectures enable to speed up system development time and to provide better performance. The design effectively utilizes the available memory of the given hardware resources. Prototype fabrication in 65-nm CMOS technology verifies high integration density and memory-utilization efficiency, good performance, and considerable flexibility in vector dimensionality, number of weight-vectors, and adaption strategies. Compared with embedded microprocessors, which rely on single-instruction-multiple-data (SIMD) processing, the developed MRPA-prototype increases the performance of both recognition and learning operations. The achieved improvements amount to approximately factors 40 and 101 on the well-established performance metrics of million connections per second (MCPS) for recognition and of million connection updates per second (MCUPS) for learning, respectively.

The prototype ASIC consumes 21.5 mW working at 150 MHz and 1 V voltage, with 2.14 mm^2 area overhead in 65nm CMOS technology. A small accuracy loss mainly comes from the truncation operation of the fixed-point operation, resulting in a quite small peak signal-to-noise ratio (PSNR) loss of 0.128 dB. The applied pipeline reconfiguration leads to a reduction in computation time and high efficiency for integration density. The applied modularity contributes to easy scalability in a both upward- and downward-compatible fashion. Additionally, the introduced shared memory-pool increases the flexibility for both the dimensionality and the number of weight vectors. Further, an implemented parameterization for system configuration adds flexibility to the choice of adaption strategies in different applications.

CHAPTER 1： Introduction

Machine vision application is under a major shift regarding the implementation and development. One of the most noticeable trends of this shift are the platforms that vision algorithms run on: from all-powerful workstations to embedded processors [1]. As is often the case, the shift origins from the intersection of market needs and available technologies. In turn, a new inter-disciplinary field has emerged from the vision community and the processor community to handle the new challenges: embedded machine vision and learning.

Over the past decades, the synergistic advances in embedded processing architectures, machine vision algorithms, integrated circuit technologies, semiconductor processes and electronic system design methodologies have increasingly expanded the application domain of embedded vision. The target market focuses on high-volume, battery-operated, cost-centric consumer applications. For example, the embedded vision techniques can help the needs for safety and security of the society. The portable platforms are well suitable for automotive safety applications, which aim to assist the driver and improve road safety.

1.1 Machine Vision and Its Embedded Applications for Mobile Devices

Human vision is incredibly fascinating and complicated. Billions of years since the evolution of our sense of sight we found that computers are on their way to matching human vision. It all started billions of years ago, where small organisms developed a mutation that made them sensitive to light. Fast forward to today, and there is an abundance of life on the planet which all have very similar visual systems. They include eyes for capturing light, receptors in the brain for accessing it, and a visual cortex for processing it. Genetically engineered and balanced pieces of a system help us do things as simple as appreciating a sunrise.

Legends are said that the machine vision began as a summer project given by Professor Marvin Minsky at Massachusetts Institute of Technology (MIT) in 1966, to an undergraduate student who is actually now a Professor at MIT, Gerald Jay Sussman. Professor Minsky said: “For this summer project, why don’t you solve the computer vision problem? You know, this really shouldn’t take too long”, and wrote an outline of what he was supposed to do. The anecdote probably have happened but in reality, the 1st machine vision project was launched by Professor Seymour Papert at MIT and given to a group of 10 students including Professor Sussman as a coordinator. The original document outlined a plan to conduct image segmentation in homogeneous backgrounds with distinct texture and color. More than half a century passed, machine vision today is far different from its definition in 1966. Plenty of topics have derived

from machine vision such as embedded vision, pedestrian detection, machine learning, and so on.

In the past 50 year, we've made even more strides to extending this amazing visual ability, not just to ourselves, but to machines as well. We've been able to closely mimic how the human eyes can. The first type of photographic camera was invented around 1816 where a small box held a piece of paper coated with silver chloride. When the shutter is open, the sliver chloride would darken where it was exposed to light. Now, 200 years later, we have much more advanced versions of the system that can capture photos right into digital form. So we've been able to closely mimic how the human eye can capture light and color. But it's turning out that was the easy part. Understanding what's in the photo is much more difficult.

The term of machine vision has not appeared in the popular media that much until recently. Part of that is because when something became successful, it got renamed. Actually, computer vision has already entered our lives. Like bar code scanning is an instance of computer vision. According to Bill Freeman, a Senior Research Scientist at Google, the computer vision researchers don't really understand how a computer see. It is like alchemy and chemistry. The alchemy came first, and then chemistry came in. Right now, we are in the alchemy stage of computer vision. Where it works, but we are not sure why. And it is the chemistry stage that we look forward to.

Consider the Fig. 1-1, our human brain can look at it and immediately know that there are pedestrians regardless the color. Our brains are cheating since we've got a couple million years worth of revolutionary context to help immediately understand what this is. But a computer doesn't have that same advantage.

Fig. 1-1. Human vision. Human brain can immediately know that there are pedestrians in the picture.

171	171	174	175	174	172	188	188
173	171	171	180	173	175	193	116
173	171	175	174	176	178	193	19
171	171	169	179	185	188	161	5
163	215	239	217	181	166	72	4
239	187	85	78	104	113	3	8
62	68	86	104	83	86	10	12
76	80	90	88	90	60	14	1

Fig. 1-2. Machine vision. There's no context here, just a massive pile of data.

From the computer perspective, the image is really just an array of numbers (Fig. 1-2), just a massive array of integer values which represent intensities across the color spectrum. There's no context here, just a massive pile of data. Or if it is color, it would be three arrays of numbers. By themselves, these pixels don't mean anything to a computer. We need to tell exactly what to do to the computer. It seems that is the computer that needs to interpret what they are. But it is not true. Computers cannot make decisions on their own. Programmers are going to build these decisions into the program, and all the computer is going to do is to reach the decision point. It turns out the context is the crux of getting algorithms to understand image content in the same way that the human brain does. And to make this work, we use an algorithm very similar to how the human brain operates using machine learning.

Machine vision is essential because it is a quite effective way to learn about the world. If we can parse what is visually around ours, we can learn a lot of information about the real world that we would not have access to otherwise.

Machine vision is the science and technology aiming to help the machines to see, representing an exciting part of cognitive and computer science. The related research includes the theory, design and implementation of algorithms that can automatically process visual data to recognize objects, track and recover their shape and spatial layout. In recent years, state-of-art advances have produced artificial systems that have reached or even surpassed human capabilities in several domains such as face detection and optical character recognition.

Machine learning helps the computers to automatically improve through experience. Machine learning allows us to effectively train the context for a data set, so that an algorithm can understand what all those numbers in a specific organization actually represent. With the machine leaning model, we can take a bunch of images of pedestrian, and as long as we feed it enough data, it will eventually be able to properly tell the difference between the two. Machine vision is taking on increasingly complex challenges and it seeing accuracy that rivals humans

performing the same image recognition tasks. But like humans, these models aren't perfect. They do sometimes make mistakes.

1.1.1 Pedestrian Detection

Pedestrian detection has always been an attractive research area among the applications of machine vision. According to the survey done by the Cisco Study [2], by 2021, roughly 80% of traffic on the internet will be video. Besides the research territory, the industry contributed enormous efforts in pedestrian recognition in various platforms such as the advanced driver assistance systems (ADAS) and integrated smart security systems market. Among the available approaches, machine vision on the embedded and portable platform is in the top tier. The current embedded platforms that rely on sophisticated algorithms have not been able to fully exploit the potential performance of machine vision algorithms, especially concerning low power consumption. Complex algorithms impose extensive computation and communication demands, requiring various stages of preprocessing, processing and machine learning blocks that need to operate concurrently. The market demands embedded platforms to operate with a power consumption of only a few watts. Efforts have been made to accelerate traditional embedded approaches by adding more powerful processors. This solution may solve the computation problem but still increases the power consumption. In this research, a coprocessor for sliding window-based pedestrian detection with multiple scales is proposed. The coprocessor realizes low power in a relatively small area.

Pedestrian detection became a popular research topic since advanced driving assistant systems (ADAS) and unmanned aerial vehicles need fast enough detection and decision making for enabling appropriate actions. The existing feature descriptors, including the Histogram of Oriented Gradients (HOG) [3], the Gradient Location-Orientation Histogram (GLOH) [4], the shape context [5], the Local Binary Pattern (LBP) [6], the Scale Invariant Feature Transform (SIFT) [7], and its successor the Speeded Up Robust Features (SURF) [8], have demonstrated their robustness in pedestrian detection applications. Subsequently, the Haar-like feature is often applied in face recognition [9] and in pedestrian detection [10]. Furthermore, due to the fast training speed, Haar-like feature was also used to extract a region of interest (ROI) for a second stage recognition with HOG. The two-stages and combinational feature descriptor achieved higher detection accuracy in ([11]-[13]) than a single feature descriptor.

The traditional software implementations involve translating the raw pixels into an integral image to construct a look-up table for speeding up the necessary calculations during feature extraction [14]. This commonly used integral image solution, taking advantage of an enormous

amount of memory resources, is mainly suitable for software applications on PCs. On the other hand, a sub-integral image offers a practical solution in hardware implementations [15].

Most state-of-the-art frameworks follow the sliding-window paradigm ([16]-[18]), which quantifies how likely it is for a window to cover a searched-for object in an image. Each window is divided into local regions (cells or blocks) for calculating feature vectors according to various strategies. Taking the popular research work in pedestrian detection [3], the detection window scans the image in a Raster manner. In fact, each window is divided into a number of sub-regions, called cells where a local feature vector is computed. Then blocks, each of which contains multiple normalized local cell features, are used to construct the window feature vector for detection by a classifier. The overlapped cells and blocks demonstrate that the sliding-window method represents an iterative process.

Meanwhile, many researchers have implemented the popular "HOG plus SVM (Support Vector Machine)" framework in hardware ([19]-[21]). Only multiplication and comparison are applied in [19] during the HOG execution for bin assignment, instead of the general complicated arc-tangent computing. Similar to the original algorithm, the cell features in a block are normalized with the L1-Sqrt-norm. Finally, a portion-wise classification is adopted to avoid the huge amount of memory for buffering all block features of a window. In [20], the gradient calculation is implemented by a relatively complicated coordinate rotation digital computer (CORDIC) solution. Then, the normalization processing is simplified by the Newton method with an approximated initial value. For classification, as also in [19], the partial SVM product is applied, but early rejection and detection are used. The difference of [21] in comparison to [19] and [20]is the improved energy efficiency due to the applied more advanced process technology. In summary, comparing to the original framework in [3], a partial classification is performed after the block-based normalization to avoid large feature buffers.

1.1.2 Battery-Operated Mobile Vision Requirements and Challenges

The mobile environment poses uniquely challenging constraints for designers of embedded computer vision systems. There are traditional issues such as size, weight, and power, which are readily evident. However, there are also other less tangible obstacles related to technology acceptance and business models that stand in the way of a successful product deployment. In this section, I describe these issues as well as other qualities desired in a mobile smart camera using vision algorithms to "see and understand" the scene. The target platform of discussion is the mobile handset, as this platform is poised to be the ubiquitous consumer device all around the world.

1.1.2.1 Power Consumption

Power dissipation is an important consumer metric for mobile handsets as it dictates the usage time (talk time, Internet use time, video playback time, audio playback time, etc.) and standby time. It is obvious that the longer the usage and standby time, the more desirable the device. At the same time, there is an opportunity to reduce the size and weight of the battery to achieve the same usage and standby time.

Mobile handsets have low-power consumption while operating (much less than desktop and laptops), and an almost negligible standby power when the device is not in use. This is evident in the drive for low power designs in the application processors ([22]-[24]). Consequently, designers should pay attention to the energy budget in the battery and not expect a computer vision algorithm to run continuously. To save power, for example, designers may consider turning off the camera module when it is not needed or lowering the frame rate when the desired performance is not needed.

1.1.2.2 Memory

In addition to computational horsepower needed by the computer vision algorithms, the designer should also consider memory bandwidth and memory allocation during early stages of the design process. These items are often considered as a design afterthought, which may cause the application to run slower than expected. This could result in poor device usability.

While still image processing consumes a small amount of bandwidth and allocated memory, video can be considerably demanding on today's memory subsystem. At the other end of the spectrum, memory subsystem design for computer vision algorithms can be extremely challenging because of the extra number of processing steps required to detect and classify objects.

1.1.2.3 Platform

In order to consider computer vision algorithms, the designer should consider how it would be integrated into the overall user experience. For example, in an operating scenario where a normal voice call is being made, the application processor may be lightly loaded, making it suitable to run other applications. In another example where the user is browsing the web, the camera-module companion chip may be lightly loaded or not used at all. It is important to make the computer vision application run seamlessly alongside existing applications. Otherwise, user acceptance would be low when the overall user experience suffers.

1.1.3 Machine Learning

In machine vision, we try to teach computers how to see, and that seeing can refer to understanding scenes, reconstructing 3D objects, recognizing objects, avoiding obstacles, helping blind people navigate. And a lot of this makes use of machine learning, and it also makes use of geometry and applied math.

Machine learning is a subfield of artificial intelligence (AI). Early AI programs typically excelled at just one thing. For example, Deep Blue could play chess at a championship level, but that's all it could do. Today we want to write one program that can solve many problems without needing to be rewritten. AlphaGo is a great example of that. But similar software can also learn to play Atari games. Machine Learning is what makes that possible. It's the study of algorithms that learn from examples and experience instead of relying on hard-coded rules. Classifier is a function that needs to be trained. It takes some data as input and assigns a label to it as output. The technique to write the classifier automatically is called supervised learning. To use supervised learning, we need to follow a recipe with a few standard steps. Step one is to collect training data. These are examples of the problem we want to solve. For example, to classify fruits, a description of the fruit as input based on features like its weight and texture is necessary. The training data is actually a table describing the features of different fruits. A good feature makes it easy to discriminate between different types of fruit. Think of these as all the examples we want the classifier to learn from. The more training data you have, the better a classifier you can create. A classifier is a box of rules, with feature as the input and labels as the output. The input and output type are always the same while there are many different types of classifier.

Before training, a classifier is just an empty box of rules. To train it, a learning algorithm is necessary. The learning algorithm is the procedure to create the box of rules, finding patterns in the training data.

Besides supervised learning, the second is known as unsupervised learning in which each training data contains the values of the attributes but does not contain the label. Unsupervised learning tries to find regularities in the unlabeled training data (such as different clusters under some metric space), infer the class labels and sometimes even the number of classes. In the unsupervised learning framework, a variety of methods and algorithms can be found in the literature. Major instances are represented by data clustering, density estimation, and dimensionality reduction. The goal of the learning process is usually defined through an objective function, where the learning schemes use the observations without prior knowledge of the class labels.

1.1.4 Application Example: Advanced Driver-Assistance Systems (ADAS)

Vision-based automotive safety systems have received considerable attention over the past decade. Such systems have advantages compared to those based on other types of sensors such as radar, because of the availability of low-cost and high-resolution cameras and abundant information contained in video images. Many automotive safety systems that used to rely on radar, laser, ultrasound, or other types of sensors now have their counterparts using cameras. However, various technical challenges exist in such systems. One of the most prominent challenges lies in running sophisticated computer vision algorithms on low-cost embedded systems at frame rate.

1.2 Platforms for Mobile Vision Applications

Machine vision is gaining momentum thanks to the improvement of the computational ability of the CPU. As one alternative, GPU gains plenty of attention in the machine vision research due to its parallel architecture. However, GPU is not able to competently handle the task-parallelism computations and suffers from limited interfaces. Moreover, the life cycle of GPU is quite short. As long as the new generation of GPU chips coming out, modifications on the code are required for re-optimization. Superior to both the CPU and GPU in power and resources aspect, ASIC allows the algorithms to deploy dedicated architecture, resulting in minimizing buffering to external memory and host memory. For machine vision on portable devices such as mobile phones, drones, and cars in daily human life, rather than CPU and GPU based solutions that consume high power and resources, the embedded implementation is anticipated to be the destination.

This technology category includes any device that executes vision algorithms or vision system control software. The following diagram shows a typical computer vision pipeline; processors are often optimized for the compute-intensive portions of the software workload.

The following examples represent distinctly different types of processor architectures for embedded vision, and each has advantages and trade-offs that depend on the workload. For this reason, many devices combine multiple processor types into a heterogeneous computing environment, often integrated into a single semiconductor component. In addition, a processor can be accelerated by dedicated hardware that improves performance on computer vision algorithms.

1.2.1 Co-processors

Coprocessor is common today to supplement the functions of primary processor (CPU). A coprocessor is a computer processor used to supplement the functions of the primary processor (the CPU). Operations performed by the coprocessor may be floating point arithmetic, graphics, signal processing, string processing, cryptography or I/O interfacing with peripheral devices. By offloading processor-intensive tasks from the main processor, coprocessors can accelerate system performance. Coprocessors allow a line of computers to be customized, so that customers who do not need the extra performance do not need to pay for it.

Coprocessors for floating-point arithmetic first appeared in desktop computers in the 1970s and became common throughout the 1980s and into the early 1990s. Early 8-bit and 16-bit processors used software to carry out floating-point arithmetic operations. Where a coprocessor was supported, floating-point calculations could be carried out many times faster. Math coprocessors were popular purchases for users of computer-aided design (CAD) software and scientific and engineering calculations. Some floating-point units, such as the AMD 9511, Intel I8231 and Weitek FPUs were treated as peripheral devices, while others such as the Intel 8087, Motorola 68881 and National 32081 were more closely integrated with the CPU.

Another form of coprocessor was a video display coprocessor, as used in the Atari 8-bit family, the Texas Instruments TI-99/4A and MSX home-computers, which were called "Video Display Controllers". The Commodore Amiga custom chipset included such a unit known as the Copper, as well as a Blitter for accelerating bitmap manipulation in memory.

As microprocessors developed, the cost of integrating the floating point arithmetic functions into the processor declined. High processor speeds also made a closely integrated coprocessor difficult to implement. Separately packaged mathematics coprocessors are now uncommon in desktop computers. The demand for a dedicated graphics coprocessor has grown, however, particularly due to an increasing demand for realistic 3D graphics in computer games.

Implementation of an algorithm in specific hardware is called co-processor. An algorithm implemented directly in the hardware, can execute it faster, because the only instruction that has to make is “execute the algorithm”. The principal reason of being faster, is that we are not tied to a general instruction set that there are more freedom to decide the way to resolve the problem. To implement the algorithm, the basic components of the hardware (logic gates) are joined to build other components more complex. To the implementation, several optimization techniques can be used, that cannot be used in the software implementation. For instance, divide the problem in parts to resolve at the time (parallelize).

1.2.1.1 Vision-Specific Processors and Cores: Application-Specific Integrated Circuit (ASIC)

ASICs are specialized, highly integrated chips tailored for specific applications or application sets. ASICs may incorporate a CPU, or use a separate CPU chip. By virtue of their specialization, ASICs for vision processing typically deliver superior cost- and energy-efficiency compared with other types of processing solutions. Among other techniques, ASICs deliver this efficiency through the use of specialized coprocessors and accelerators. And, because ASICs are by definition focused on a specific application, they are usually provided with extensive associated software. This same specialization, however, means that an ASIC designed for vision is typically not suitable for other applications. ASICs' unique architectures can also make programming them more difficult than with other kinds of processors; some ASICs are not user-programmable.

If we implement an algorithm in a chip we were talking about a specific purpose processor. For instance, image a processor that only have the instruction "sum two numbers". If we have to multiply two numbers with the instruction, the processor will have to execute it several times to get the result. Instead, if we have implemented hardware that allows multiply directly, we only would have to execute the multiply instruction once. In conclusions, when more complex is the instruction to implement, we will save time if we implement it in hardware. Some well-established products and highly publicized technologies may be seen as early examples of embedded machine vision. Two examples are the optical mouse (which uses a hardware implementation of an optical flow algorithm), and NASA's Martian rovers, Spirit and Opportunity (which used computer vision on a processor of very limited capabilities during the landing, and which have a capability for vision-based self-navigation).

1.2.1.2 Field-Programmable Gate Array (FPGA)

Instead of incurring the high cost and long lead-times for a custom ASIC to accelerate computer vision systems, designers can implement an FPGA to offer a reprogrammable solution for hardware acceleration. With millions of programmable gates, hundreds of I/O pins, and compute performance in the trillions of multiply-accumulates/sec (tera-MACs), high-end FPGAs offer the potential for highest performance in a vision system. Unlike a CPU, which has to time-slice or multi-thread tasks as they compete for compute resources, an FPGA has the advantage of being able to simultaneously accelerate multiple portions of a computer vision pipeline. Since the parallel nature of FPGAs offers so much advantage for accelerating computer vision, many of the algorithms are available as optimized libraries from

semiconductor vendors. These computer vision libraries also include preconfigured interface blocks for connecting to other vision devices, such as IP cameras.

1.2.2 General-Purpose CPUs

The software always has to be executed in the hardware of the machine where resides. Normally, always we have a general purpose processor, the name is due to is built to execute any algorithm,

While computer vision algorithms can run on most general-purpose CPUs, desktop processors may not meet the design constraints of some systems. However, x86 processors and system boards can leverage the PC infrastructure for low-cost hardware and broadly-supported software development tools. Several Alliance Member companies also offer devices that integrate a RISC CPU core. A general-purpose CPU is best suited for heuristics, complex decision-making, network access, user interface, storage management, and overall control. A general purpose CPU may be paired with a vision-specialized device for better performance on pixel-level processing.

1.2.3 Graphics Processing Units (GPU)

High-performance GPUs deliver massive amounts of parallel computing potential, and graphics processors can be used to accelerate the portions of the computer vision pipeline that perform parallel processing on pixel data. While General Purpose GPUs (GPGPUs) have primarily been used for high-performance computing (HPC), even mobile graphics processors and integrated graphics cores are gaining GPGPU capability-meeting the power constraints for a wider range of vision applications. In designs that require 3D processing in addition to embedded vision, a GPU will already be part of the system and can be used to assist a general-purpose CPU with many computer vision algorithms. Many examples exist of x86-based embedded systems with discrete GPGPUs.

1.3 Contributions: Power-Efficient Acceleration Coprocessors for Embedded Vision

As architectures evolve towards multi-cores composed of a mix of cores and accelerators, a machine learning accelerator is able to reach the rare combination of efficiency and application flexibility. Most of the previous machine-learning accelerators focused on efficient implementation of the computational part of the algorithms, paying little attention on the usage of memory. As a result, recent state-of-the-art accelerators are characterized by their large size.

In this study, we design two accelerators. One is for HOG-based inference, utilizing a cell-based image-scan strategy. The original HOG algorithm is implemented in a further improved hardware-oriented way, applying a cell-based scan manner. A powerful cell-feature reuse method is employed to improve the performance by vastly reducing the number of the cell-features that must be repetitively calculated in the original HOG algorithm, which applies a block-based scan manner. Furthermore, the developed cell-based scan manner drastically reduces the memory requirements, since the need to buffer whole images is eliminated, and adds processing flexibility of input images with in principle unlimited height. Faster speed and less area consumption are achieved in comparison to previous approaches, because we employ fewer multiplication operations without sacrificing accuracy.

The other is for LVQ-based learning, with a special emphasis on the impact of memory on accelerator design, performance and energy. The developed modular and reconfigurable pipeline-architecture (MRPA) for LVQ neural networks has the following advantages. First, the MRPA accelerates the computational speed and provides high integration density by the implementation of pipeline reconfiguration. All the weight-vectors share the same arithmetic and logic units (ALUs), rather than having individual ALUs. Meanwhile, the MRPA improves the memory-utilization efficiency by segregating the weight-memory blocks from the processing elements (PEs) as a shared memory pool. The memory sharing scheme also increases the flexibility of the weight-vector, in contrast to the SIMD methods, which directly map neurons to PEs. The size of an individual weight-memory block and the number of weight-memory blocks limit the range of manageable dimensionality and number of weight-vectors. The MRPA overcomes the limitation. Both the dimensionality and number of weight-vectors are adaptable to a wide range of applications. Moreover, the modularity of the design in the MRPA leads to easy scalability for future soft-IP design.

Reference

[1] Kisacanin, B., Bhattacharyya, S. S., & Chai, S. (Eds.). Embedded computer vision. Springer Science & Business Media, 2008.

[2] https://tubularinsights.com/video-2021/

[3] N. Dalal, and B. Triggs, “Histograms of oriented gradients for human detection,” IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol.1, pp.886-893, 2005.

[4] K. Mikolajczyk, and C. Schmid, “A performance evaluation of local descriptors,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.27, no.10, pp. 1615-1630, Oct. 2005.

[5] S. Belongie, J. Malik, and J. Puzicha, “Shape matching and object recognition using shape contexts,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no.4, pp.509-522, April 2002.

[6] T. Ojala, M. Pietikäinen, and T. Mäenpää, “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, July 2002.

[7] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” International journal of computer vision, vol.60, no.2 pp. 91-110, 2004.

[8] H. Bay, T. Tuytelaars, and L. V. Gool, “Surf: Speeded up robust features,” Computer vision–ECCV2006, pp. 404-417, May 2006.

[9] P. Viola and M. J. Jones. Robust real-time face detection. International Journal of Computer Vision, 57(2), pp. 137–154, 2004.

[10] S. Zhang，C. Bauckhage，AB. Cremers, “Informed Haar-Like Features Improve Pedestrian Detection”, CVPR, 2014.

[11] X. Yuan，X. Shan，L. Su, “A Combined Pedestrian Detection Method Based on Haar-Like Features and HOG Features”, International Workshop on Intelligent Systems & Applications, 2011.

[12] Y. Wei，Q. Tian，T. Guo, “An Improved Pedestrian Detection Algorithm Integrating Haar-Like Features and HOG Descriptors”, Advances in Mechanical Engineering, 2013.

[13] Y. Li，W. Lu，S. Wang，X. Ding, “Pedestrian Detection Using Coarse-to-Fine Method with Haar-Like and Shapelet Features”, International Conference on Multimedia Technology, 2010.

[14] P. Viola and M. Jones, "Rapid object detection using aboosted cascade of simple features," IEEE Conference on Computer Vision and Pattern Recognition, pp. 511-518, vol.1, 2001.

[15] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, FPGA-based face detection system using Haar classifiers, In Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, pp.103-112, 2009.

[16] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.32, no.9, pp.1627-1645, Sep. 2010.

[17] C. H. Lampert, M. B. Blaschko, and T. Hofmann, “Beyond sliding windows: Object localization by efficient subwindow search,” IEEE Conference on Computer Vision and Pattern Recognition, pp.1-8, June 2008.

[18] B. Alexe, T. Deselaers, and V. Ferrari, “Measuring the objectness of image windows,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.34, no.11, pp.2189-2202, Nov. 2012.

[19] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, K. Doll, FPGA-Based Real-Time Pedestrian Detection on High-Resolution Images. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 629-635, 2013.

[20] K. Mizuno, K. Takagi, S. Izumi, H. Kawaguchi, 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. E96-C (4), pp. 433-443, 2013.

[21] 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 (3), pp. 325-337, 2016.
[22] D. Talla, J. Gobton, "Using DaVinci technology for digital video devices," *Computer,* v. 40, no.10, Oct. 2007, pp. 53-61.
[23] Max Baron, "Freescale's MXC voted best: the crown goes to Freescale's MXC91321 chip," *Microprocessor Report,* January 30, 2006, pp. 1-3.
[24] A. Bellaouar, M. I. Elmasry, *Low-Power Digital VLSI Design: Circuits and Systems,* Springer, June 30, 1995.

CHAPTER 2: Theoretical Basis of Pedestrian Detection in Mobile Devices

2.1 Classic Methods: Feature Based Detection

2.1.1 Feature Extraction

Good features are informative, independent, and simple. Classifiers are only as good as the features provided. That means coming up with good features is one of the most important jobs in machine learning. But what makes a good feature, and how can we tell. For a binary classification, a good feature makes it easy to decide between two different things. There's a lot of variation in the world. So when thinking of a feature, it is necessary to consider how it looks for different values in a population. That's why machine learning needs multiple features. In machine vision applications, feature vectors are used to represent the perceived environment. Relational descriptions are of crucial importance in high-level vision. Feature descriptor helps reduce the search space of the classifiers by modeling high-dimensional data as a combination of a few active features and, hence, can reduce the computation required for classification. In general, these methods can be divided into two strategies. One is part-based approaches ([1]-[3]), which utilize individual detectors to locate single parts. The recognition results depend on whether the detected parts are arranged in a geometrically plausible configuration. The other one refers to holistic approaches which shift detection or scan windows over the image with dense positions and scales ([4] and [5]).

Histograms of oriented gradient (HOG) descriptors are based on the sliding-window strategy, and are proved to be competitive in terms of classification performance in a large variety of recognition tasks and to have robustness against illumination changes, by recent experimental studies ([6]and [7]). Since the original HOG feature is highly-dimensional (3,780 elements per search window), the real-time operation is a fundamental problem in the HOG descriptor research field. Reference [8] reported an object-detection system composed of a HOG descriptor and a support vector machine (SVM) as classifier, which needs about one second to process a QVGA image with 320×240 pixels. According to Ref. [8], the runtime for a VGA-image with 640×480 pixels is 13.3 s in their benchmark report.

2.1.2 Trained Classification

Machine learning is one of most rapidly growing technical fields, lying at the core of data science and artificial intelligence, and at the intersection of statistics and computer science. Recent progress in machine learning has originated from two sources. The first one is the

development of new learning algorithms and theory. The other one is exploration of the low-cost computation. There have been two fundamentally different types of tasks in machine learning, supervised learning and unsupervised learning. Both of them emerge as the most important learning strategy.

2.2 Deep Learning-Based Methods

In the last few years, deep neural networks have led to breakthrough results on a variety of pattern recognition problems, such as computer vision and voice recognition. One of the essential components leading to these results has been a special kind of neural network called a convolutional neural network.

At its most basic, convolutional neural networks can be thought of as a kind of neural network that uses many identical copies of the same neuron. It should be noted that not all neural networks that use multiple copies of the same neuron are convolutional neural networks. Convolutional neural networks are just one type of neural network that uses the more general trick, weight-tying. Other kinds of neural network that do this are recurrent neural networks and recursive neural networks. This allows the network to have lots of neurons and express computationally large models while keeping the number of actual parameters, the values describing how neurons behave, that need to be learned fairly small.

The specific type of neural network that accomplishes recognition is called a convolutional neural network or CNN. CNNs work by breaking an image down into smaller groups of pixels called filter. Each filter is a matrix of pixels, and the network does a series of calculations of these pixels comparing them against pixels in a specific pattern the network is looking for. In the first layer of CNN, it is able to detect high-level patterns like rough edges, and curves. As the network performs more convolutions, it can begin to identify specific objects like faces and animals. How does a CNN know what to look for and if its prediction is accurate? This is done through a large amount of labeled training data. When the CNN starts, all of the filters values are randomized. As a result, its initial predictions make little sense. Each time the CNN makes a prediction against a labeled data, it uses an error function to compare how close its prediction was to the image's actual label. Based on this error or loss function, the CNN updates its feature values and starts the process again. Ideally, each iteration performs with slightly more accuracy.

What if instead of analyzing a single image, we want to analyze a video using machine learning. At its core, a video is just a series of image frames. To analyze a video, we can build on our CNN for image analysis. In still images, we can use CNNs to identify features, but when we move to videos, things get more difficult since the items we're identifying might change over

time. Or, more likely, there's context between the video frames that's highly important to labeling. For example, if there's a picture of a half full cardboard box, we might want to label it packing a box or unpacking a box depending on the frames before and after it. This is where CNNs come up lacking. They can only take into account spatial features, the visual data in an image, but can't handle temporal or time features: how a frame is related to the one before it. The address this issue, researchers have taken the output of CNN and feed it into another model which can handle the temporal nature of videos. This type of model is called a recurrent neural network (RNN).

2.3 Challenges and Limitations for Mobile Devices

Mobile devices can be separated into three categories: low-cost, mid-tier, and smart phones. The lower-end phones are low-cost, high-volume devices with little to no features, except the ability to make phone calls. These low-cost phones may not even include a camera. The mid-tier phones are mid-range in prices with standard features such as a megapixel camera. They may be targeted toward the teens and tweens, and may have a music player for music enthusiasts.

Smart phones offer advanced capabilities much like a portable computer. These phones are decked out with features such as PDA functions, large displays, and high-resolution (multi-megapixel) camera. They are targeted to the tech-savvy and also the business professionals, that is, those who can afford the premium cost. New computer vision applications are likely to first appear in smart phones. These mobile handsets have higher performance computing platform that can handle the extra application load. Furthermore, the better cameras provide better resolution and higher quality images for computer vision algorithms.

Designers of computer vision algorithms should consider the psychological aspects that influence the acceptance of the technology. With understanding of how users effectively use computer vision features in a mobile handset and what motivates them to continue using the feature, designers can make inroads into having the technology as a commonplace feature set.

Furthermore, we should consider the sociological impact of pervasive computer vision technology in our everyday lives. This statement is not necessarily a call to monitor and examine every aspect of the technology in our society. Instead, it is an opinion for designers to consider computer vision applications that have great social impact. Technology such as computer vision can be applied to improve daily lives by making routine tasks faster and safer. We should seek to utilize mobile technology to improve the way to communicate and educate ourselves.

References

[1] P. F. Felzenszwalb and D.P. Huttenlocher, Int. J. Comput. Vision 61, 55 (2005).

[2] S. Ioffe and D.A. Forsyth, Int. J. Comput. Vision 43, 45 (2001).

[3] K. Mikolajczyk, C. Schmid and A. Zisserman, Proc. IEEE European Conf. Computer Vision, 2004, p. 69.

[4] A. Tejani, D. Tang, R. Kouskouridas and T. K. Kim, European Conf. Computer Vision, 2014, p. 462.

[5] L. A. Jeni, J. F. Cohn and T. Kanade, 11th IEEE Int. Conf. Workshop Automatic Face and Gesture Recognition 2015, vol. 1, p. 1.

[6] P. Dollár, C. Wojek, B. Schiele and P. Perona, Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2009, p. 304.

[7] M. Enzweiler and D. M. Gavrila, IEEE Trans. Pattern Anal. Mach. Intell. 31, 2179 (2008).

[8] N. Dalal, and B. Triggs, "Histograms of oriented gradients for human detection," IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol.1, pp.886-893, 2005.

CHAPTER 3: Pedestrian Detection Coprocessors

Pedestrian detection became a popular research topic since advanced driving assistant systems (ADAS) and unmanned aerial vehicles need fast enough detection and decision making for enabling appropriate actions. This chapter reports an algorithm and hardware co-design approach to enable real-time and energy-efficient pedestrian-detection accelerators, which can process XGA (1024 × 768, 30 fps) video in real time, achieving 50 MHz feature extraction and 200 MHz classification, with energy consumption of 906 pJ/pixel. The histogram of oriented gradients (HOG) algorithm for feature extraction, which is known to provide high efficiency and accuracy is implemented in 180 nm CMOS technology. The cell-based processing decomposes the high dimensional window feature-vector and reduces the overhead of parallel multi-window detection by reuse of the cell features, avoiding many of the massive and repetitive calculations in a conventional block-based algorithm. Also, an effective bin decoder is combined with the orientation calculation in the applied HOG feature-extraction algorithm. A parallelized voting element (PVE) ensures an efficient pipelined histogram calculation with dual-port memories. Detection accuracy can be improved using complementary features in addition to the HOG feature, at the cost of an extra 40% power consumption, 64% area requirement, and 53% memory size.

3.1 Overview for Histogram of Oriented Gradient (HOG) Feature Extractor

Intelligent computer vision systems have been presented with numerous solutions for platforms such as robotics, automotive systems, security systems, mobile devices, and wearable electronics. However, practical and robust solutions for numerous applications are still very challenging in the case of required real-time performance due to various constraints, e.g. computational power or battery capacity. Object recognition in images has been a very active filed of research, and attracts an enormous amount of research interest ([1]-[3]). It is still a challenge due to the wide variability in appearance of outdoor scenes, such as complex backgrounds[4], different poses[5] and illumination conditions[6]. To solve these challenges, a wide variety of methods have emerged in terms of algorithm and architecture developments for feature extraction ([7]-[10]) as well as classification ([11]-[14]). In many cases, the typical low power processor used for portable electronics or robotics platforms has the insufficient computational power to implement complex machine-learning applications for meeting real-time constraints. One effective methodology for solving detection and recognition problems

while meeting these constraints is through dedicated hardware-based approaches. Even though hardware-architecture development may involve undesirable complexity, flexibility, and accuracy trade-offs for intelligent and computer vision systems, compared to purely software-based systems, hardware architectures are capable of meeting the usually tough real-time and low-power constraints.

A challenging and important issue for object recognition is feature extraction on embedded systems. In general, these methods can be divided into two strategies. One is part-based approaches ([15]-[17]), which utilize individual detectors to locate single parts. The recognition results depend on whether the detected parts are arranged in a geometrically plausible configuration. The other one refers to holistic approaches which shift detection or scan windows over the image with dense positions and scales ([18][19]). Histograms of oriented gradient (HOG) descriptors are based on the sliding-window strategy, and are proved to be competitive in terms of classification performance in a large variety of recognition tasks and to have robustness against illumination changes, by recent experimental studies ([20][21]).

3.1.1 Original Software-Based HOG Algorithm and Previous Work

The existing feature descriptors, including the HOG [22], the Gradient Location-Orientation Histogram (GLOH) [23], the shape context [24], the Local Binary Pattern (LBP) [25], the Scale Invariant Feature Transform (SIFT) [26], and its successor the Speeded Up Robust Features (SURF) [27], have demonstrated their robustness in pedestrian detection applications. Subsequently, the Haar-like feature is often applied in face recognition [28] and in pedestrian detection [29]. Furthermore, due to the fast training speed, Haar-like feature was also used to extract a region of interest (ROI) for a second stage recognition with HOG. The two-stages and combinational feature descriptor achieved higher detection accuracy in ([30]-[32]) than a single feature descriptor.

The traditional software implementations involve translating the raw pixels into an integral image to construct a look-up table for speeding up the necessary calculations during feature extraction [33]. This commonly used integral image solution, taking advantage of an enormous amount of memory resources, is mainly suitable for software applications on PCs. On the other hand, a sub-integral image offers a practical solution in hardware implementations [34].

Most state-of-the-art frameworks follow the sliding-window paradigm ([35]-[37]), which quantifies how likely it is for a window to cover a searched-for object in an image. Each window is divided into local regions (cells or blocks) for calculating feature vectors according to various strategies. Taking the popular research work in pedestrian detection [22], the

detection window scans the image in a Raster manner. In fact, each window is divided into a number of sub-regions, called cells where a local feature vector is computed. Then blocks, each of which contains multiple normalized local cell features, are used to construct the window feature vector for detection by a classifier. The overlapped cells and blocks demonstrate that the sliding-window method represents an iterative process.

Meanwhile, many researchers have implemented the popular "HOG plus SVM (Support Vector Machine)" framework in hardware ([38]-[49]). Only multiplication and comparison are applied in [38] during the HOG execution for bin assignment, instead of the general complicated arc-tangent computing. Similar to the original algorithm, the cell features in a block are normalized with the L1-Sqrt-norm. Finally, a portion-wise classification is adopted to avoid the huge amount of memory for buffering all block features of a window. In [39], the gradient calculation is implemented by a relatively complicated coordinate rotation digital computer (CORDIC) solution. Then, the normalization processing is simplified by the Newton method with an approximated initial value. For classification, as also in [38], the partial SVM product is applied, but early rejection and detection are used. The difference of [49] in comparison to [38] and [39] is the improved energy efficiency due to the applied more advanced process technology. In summary, comparing to the original framework in [22], a partial classification is performed after the block-based normalization to avoid large feature buffers.

Since the original HOG feature is highly-dimensional (3,780 elements per search window), the real-time operation is a fundamental problem in the HOG descriptor research field. However, the conventional descriptor implementations cannot process the inputted pixels immediately, which means they are not suitable for real-time processing. Only after the whole image is scanned and buffered, these descriptor implementations can begin to extract features. Reference [22] reported an object-detection system composed of a HOG descriptor and a support vector machine (SVM) as classifier, which needs about one second to process a QVGA image with 320×240 pixels. According to Ref. [22], the runtime for a VGA-image with 640×480 pixels is 13.3 s in their benchmark report. There have been research efforts, attempting to accelerate the HOG feature construction based on software as well as hardware techniques. In Ref. [40], an integral map (IMAP) is utilized to speed up the HOG-descriptor extraction, and a cascade-of-rejecters approach is employed instead of SVM. Besides the pure software approaches, Ref. [41] presented an implementation based on graphic processing units (GPU). Reference [42] took advantage of FPGAs, and reported a simplified HOG implementation, which is able to process 60 frames of 752×480-pixel images per second on a Xilinx Virtex-4 FPGA with relatively high power consumption. In other words, the previously

obtained results verified that traditional software- and hardware-based implementations are insufficient for real-time mobile applications.

The original HOG algorithm was firstly introduced by Dalal in 2005 [22], to extract a feature set from an image for object recognition tasks. The HOG descriptor characterizes local object shape and appearance by the distribution of local intensity gradients, instead of the precise knowledge of the corresponding gradient [43].

The default scan window for detection covers 64×128 pixels. The basic element for feature construction is called a cell and has 8×8 pixels. Thus a detection window is divided into 8×16=128 cells, and overlaps 7×15 blocks across a 64×128 pixel detection window. First of all, the pixel gradient in horizontal G_x and vertical G_y direction is computed. Given a normalized input image, the HOG descriptor extraction begins with the computation of the vertical and horizontal gradients for every pixel in the input image. Simple one-dimensional [-1 0 1] vertical and horizontal gradient masks are applied to each pixel for computing the gradients. Then, gradient orientation and magnitude according to Eqs. (3-1) and (3-2) are calculated for each pixel. The next step is the orientation binning. The magnitude in (3-3) is an approximation of the original expression (3-2) contributing to the histogram distribution. The orientation bins of the histogram are based on a specified number of orientation divisions according to (3-1), ranging between $-90°$ to $90°$. Here, the variables $M(i,j)$ and $O(i,j)$ represent the magnitude and orientation of a pixel at positions i and j. The variables $G_x(i,j)$ and $G_y(i,j)$ are the vertical and horizontal gradients, respectively.

$$O(i,\ j) = tan^{-1}\left(\frac{G_y(i,j)}{G_x(i,j)}\right) + 90^o \tag{3-1}$$

$$M(i,\ j) = \sqrt{{G_x}^2(i,\ j) + {G_y}^2(i,\ j)} \tag{3-2}$$

$$M(i,j) = |G_x(i,j)| + |G_y(i,j)| \tag{3-3}$$

The gradient angles vary between 0 and 180°, and the range is evenly divided into nine bins covering 20° each, e.g., 0 to 20°. A histogram of gradient orientations in an image cell is computed by taking into account the contributions of all pixels in the cell. A nine-bin histogram for each cell is generated by accumulating weighted gradient magnitudes in each of the bins. In other words, the extracted information is compressed into a nine-dimensional vector for each cell. Groups of 2×2 adjacent cells are known as blocks, and adjacent blocks are defined to have horizontal and vertical overlaps of one cell. This results in 7×15=105 blocks for a detection window. The characteristic feature of a block is formed by concatenating the 4 cell histograms within the block, resulting in a 9×4=36-dimensional feature vector. The HOG descriptor of a

detection window is represented by a concatenation of all these block features. Consequently, the HOG descriptor of a detection window is represented by a vector with 105×36=3780 dimensions. The magnitude (3-3) is the weighted vote contribution of the histogram for each pixel within the cell.

The arc-tangent function is of high complexity and very expensive to calculate, especially for a hardware implementation. As a result, hardware-friendly approximation algorithms are essential. Most of these approximation algorithms either utilize an iterative architecture at the cost of system deceleration, e.g. CORDIC, or a lookup-table (LUT) method resulting in critical requirements of memory. In [38], the angle computation is replaced by simple integer multiplications and logical comparisons to determine the bin assignment (gradient orientation). Dalal et al. in [22] have analyzed the influence of the bin number on performance and claimed that the best performance is achieved for 9 bins.

For the HOG descriptor, the feature extraction sub-component uses a cell size of 8×8 pixels, a block size of 2×2 cells, and overlapping 7×15 blocks across a 64×128 pixel detection window. A final feature-vector size of 3780 dimensions is obtained.

3.1.2 Hardware-Oriented Algorithm for HOG: Cell-Based

In this research, the original HOG algorithm is implemented in a further improved hardware-oriented way, applying a cell-based scan manner. A powerful cell-feature reuse method is employed to improve the performance by vastly reducing the number of the cell-features that must be repetitively calculated in the original HOG algorithm, which applies a block-based scan manner. Furthermore, the developed cell-based scan manner drastically reduces the memory requirements, since the need to buffer whole images is eliminated, and adds processing flexibility of input images with in principle unlimited height. In addition, an effective bin decoder is combined with the orientation calculation. One of the most important aspects of the proposed bin decoder comes from the capability of estimating the bin for a cell with less time as well as resources. Furthermore, faster speed and less area consumption are achieved in comparison to previous approaches ([42][44]), because we employ fewer multiplication operations without sacrificing accuracy. Then, the parallelized voting element (PVE) ensures a pipelined histogram calculation with dual-port memories. The proposed architecture with high hardware efficiency enables real-time speed, high processing-flexibility of different image sizes, and low area- as well as power consumption. To outline the resource efficiency of the proposed algorithm and architecture, we present the practical realization as a test chip

fabricated in 180 nm CMOS technology. We evaluate the high performance of this prototype chip by using XGA (1024×768 pixels) size images.

3.1.2.1 Feature Extraction Scheme

Within each sliding window, block shifting by one cell in horizontal and vertical direction results in 7×15 = 105 overlapping blocks. For each of these blocks, a normalization step is applied to adjust the contained-cell descriptors during construction of the block's local FV. As shown in Fig. 3-2. **Cell-based feature extraction. The map for the reuse times of each cell in a sliding window (64 × 128 pixels) are summarized in relation to cell position. Sliding windows are shifted in block units (2×2 cells) during image-recognition processing.**, block 1 (B1) and block 2 (B2) have two overlapped cells. In other words, these two overlapped cells are reused two times. Furthermore, the presence of e.g. cell (2, 2) in four blocks (B1, B2, B8, and B9) leads in a 4-fold reuse of this cell. Consequently, depending on the cell positions in a window, the reuse time of each cell can be deduced. The complete map of the reuse time of cells (MRToC) in a sliding window for the descriptor vector construction is summarized in Fig. 3-2. The FV of the j-th sliding window (w_j) is constructed according to (3-4) where t_i is the reuse time of the i-th cell belonging to w_j and d_{cell} is the dimensionality of the local FV of each cell. For the case of a window with 64×128 pixels, the dimensionality of the HOG descriptor is $((\mathbf{1}\times 4+\mathbf{2}\times 40+\mathbf{4}\times 84)\times 9=3780)$. Instead of a block-based HOG-descriptor extraction, the proposed cell-based scan method, which exploits the cell-overlap characteristics of blocks for cell-

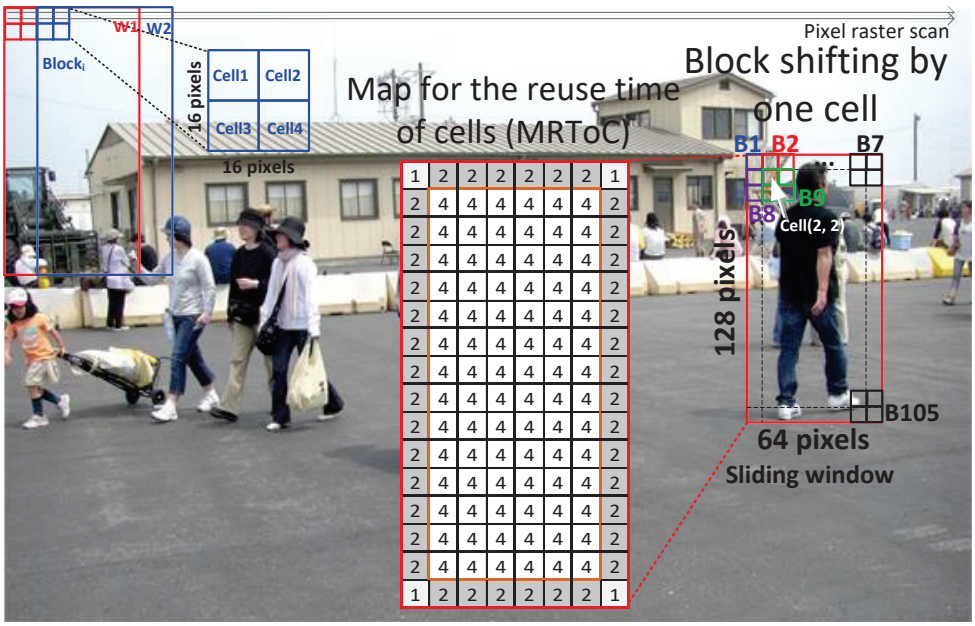


Fig. 3-1. Cell-based feature extraction. The map for the reuse times of each cell in a sliding window (64 × 128 pixels) are summarized in relation to cell position. Sliding windows are shifted in block units (2×2 cells) during image-recognition processing.

histogram reuse in multiple block histograms, needs to calculate the feature vector of each cell only once. Therefore, the proposed cell-based method in a raster-scan manner without normalization reduces the construction time of the HOG-descriptor vector by a factor of 3.28 (420/128) in comparison to the block-based method.

$$D_{FV} = \sum_{t_{i \in w_j}} (t_i \times d_{cell}) \tag{3-4}$$

In addition, the cell-based scan method immediately processes each pixel data, enabling synchronized processing with the image sensor. This means, the frame or integral-image buffers are not necessary for this implementation. Furthermore, because of the prompt processing, the histogram memory has to store only x/8 intermediate partial descriptor vectors of one cell row in case of an input image with x×y pixels. After the HOG-feature calculation of x/8 cells is completed, the corresponding memory locations can be overwritten by the intermediate data for the next cell row. This leads to the processing flexibility of input images with in principle unlimited height.

Typically, integral images are used in software implementations, exploiting a large amount of DRAM available in personal computers (PCs) [29]. The hardware implementations often store an integral image for a sub-image in addition to the image-frame buffer [34]. In case of integral-image usage for the entire original image, the bit-width of each word for integral pixels requires $\lceil \log_2(w \times h \times I(x)) \rceil$-bit, where w is the image width, h is the image height, and $I(x)$ is the image-pixel value. For example, each word of the integral image for an original VGA gray-scale image has a bit-width of 27-bit. On the other hand, for a sub-image with 16×16 pixels, the bit-width of integral image words can be reduced to 16-bit.

Without an integral image buffer (at least 4 Kb) and an image-frame buffer, I propose a cell-based feature extraction circuitry as illustrated in Fig. 3-2 with pixel-based pipelined architecture relying on the pixel-transfer frequency of the image sensor. The sizes of a window and its cells are depended on different target objects in the image. In this paper, we define a fixed size for the sliding window with 64×128 pixels and the cell with $(C_{height} \times C_{width})$ pixels. In the case of a cell with 8×8 pixels, a $w \times h$-pixel input image has $w/8$ cells in horizontal direction. Each seriatim-input pixel from the image sensor is immediately processed for feature descriptor calculation. Intermediate calculation results are then temporarily saved in a storage unit. Once the last pixel of a cell (e.g., pixel p[7w+7] of the first cell in the first line of cells) is processed, the calculation of the descriptor for this cell is completed and the result can be transferred to the recognition unit. During the input of the last pixel line (p[7w+i], $i \in$[0,w-1]) of cell lines, the local FV $\boldsymbol{v}$ for a cell can be completed in every 8th clock cycle.

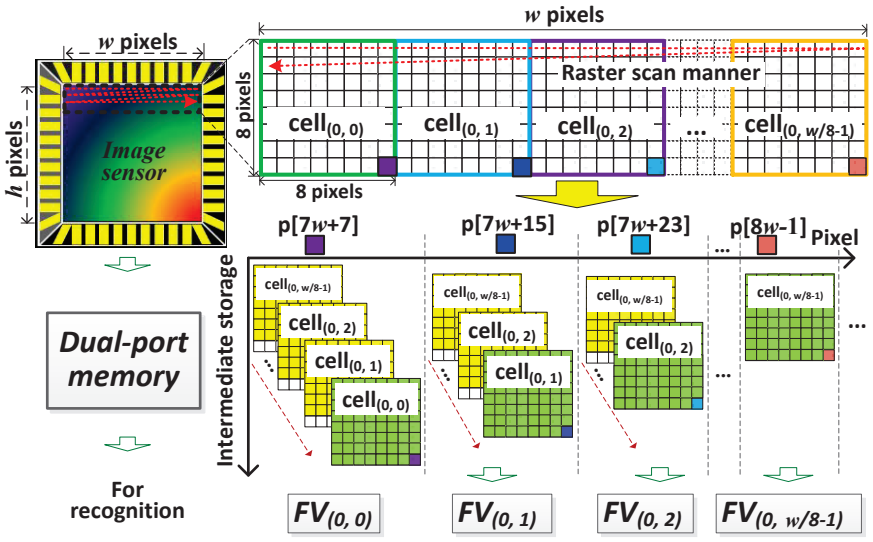


Fig. 3-3. Cell-feature extraction by a pixel-based pipeline architecture.

Accordingly, the one-row cell features take $w \times (C_{height} - 1) + \frac{w}{C_{width}} \cdot C_{height} + \tau$ clock cycles where the pipeline delay of the feature extraction circuit is τ. Eventually, the speed for extracting one cell is $\frac{w \times (C_{height} - 1) + \frac{w}{C_{width}} \cdot C_{height} + \tau}{\frac{w}{C_{width}}} \times f \; s$ where f is clock frequency of the image sensor.

An essential limitation of many hardware implementations for feature extraction is the critical requirements for memory space. In practice, different applications have different image sizes. To limit the necessity of excessive on-chip memory for ASIC implementations or of large internal SRAM blocks for FPGA implementations, an overwrite concept of obsolete intermediate results is proposed for optimal utilization of the embedded SRAM. The key feature of the concept is the capability to reuse the SRAM for intermediate cell FVs by overwriting obsolete results whenever possible so that memory-utilization efficiency and at the same time application flexibility can be significantly improved. The memory usage is quantitatively illustrated in (3-5) where C_{width} is the width of a cell and d_{cell} is the dimensionality of a cell. In this work, each cell of the HOG feature has 9-d vector components ($d_{cell} = 9$). In addition, the bit precision of each vector component is 16-bit. According to (3-5), the memory usage solely depends on the image width *w*. The proposed pixel-based pipeline architecture for cell feature extraction eliminate the influence of the image height *h* on memory consumption by the immediately processing of each pixel and the overwriting of the obsolete intermediate results. Furthermore, both the image width and the image height have a wide range of adjustability in this work. The proposed architecture is generalizable and can be useful in a wide range of applications.

$$Mem_{FV} = \frac{w}{C_{width}} \times d_{cell} \times 16\ bit \quad (3\text{-}5)$$

Comparing to the straightforward hardware implementation, besides a frame buffer with $w \times h \times 8$ bits, they need a memory to store the window-based feature with 3780×16 bits for the case of the HOG feature. Whereas, in this work, only $18w$-bit memories are required for the HOG feature extraction.

3.1.2.2 Recognition in Sliding Window Paradigm

In the literature ([35]-[37]), the sliding window is scanned across the image to all positions on the defined grid of overlapping blocks. Usually, a very large buffer stores the determined window FVs with several thousand components each for the entire image. In general, windows are overlapped in block unit (e.g. 2×2 cells, W1 and W2 in Fig. 3-1).

In this research, I propose the parallel recognition processing for each cell in all related overlapping sliding windows (OSWs). Within this proposal, the recognition process can be executed in parallel for all OSWs including a given cell. In the proposed cell-based recognition, the cell has a different position in each OSW according to the MRToC. Given an input image with $w \times h$ pixels, an OSW with $W_{width} \times W_{height}$ pixels, a cell with $C_{width} \times C_{height}$ pixels, and a block with 2×2 cells, the number of the OSWs to which the cell belongs can be derived from the cell position in the image C[c, r] ($0 \leq c \leq \frac{w}{C_{width}}$, $0 \leq r \leq \frac{h}{C_{height}}$). Initially, the first window (FW) containing the current cell C[c, r], should be located. The numbers of OSWs containing C[c, r] in horizontal and vertical directions (WN_{hor} and WN_{ver}) are then calculated. The index of the FW ($index_{FW}$) with one-dimensional numbering, WN_{hor} and WN_{ver} are derived from (3-6), (3-7) and (3-8), respectively, while c and r with $0 \leq c \leq \frac{w}{C_{width}}$ and $0 \leq r \leq \frac{h}{C_{height}}$ are the indices of a cell. Furthermore, the maximum for WN_{hor} is $\frac{W_{width}}{2 \times C_{width}}$ and the maximum for WN_{ver} is $\frac{W_{height}}{2 \times C_{height}}$. In the case of an OSW with 64×128 pixels and a cell with 8×8 pixels, the maximum WN_{hor} is 4 and the maximum WN_{ver} is 8. In other words, one cell can be included in up to $4 \times 8 = 32$ OSWs.

We take the example of cell C[15, 41] in a VGA (640 × 480 pixels) image. In the beginning, the *index* of the FW is computed through (3-6). Since in this example $c \geq \frac{W_{width}}{C_{width}}$ and $r \geq \frac{W_{height}}{C_{height}}$, i.e. $15 \geq 8$ and $41 \geq 16$, the result $index_{FW} = 485$ is determined. Then, $WN_{hor} = 4$ and $WN_{ver} = 8$ are calculated according to (3-7) and (3-8). Consequently, the 485th OSW

is taken as the origin and a partial recognition procedure is performed for a 4×8 matrix of OSWs.

$$index_{FW} = \begin{cases} 0 & c < \frac{W_{width}}{C_{width}}, r < \frac{W_{height}}{C_{height}} \\ \left\lceil \frac{c - \frac{W_{width}}{C_{width}} + 1}{2} \right\rceil & c \geq \frac{W_{width}}{C_{width}}, r < \frac{W_{height}}{C_{height}} \\ \left\lceil \frac{r - \frac{W_{height}}{C_{height}} + 1}{2} \right\rceil \cdot \left(\frac{w - W_{width}}{2 \times C_{width}} + 1 \right) & c < \frac{W_{width}}{C_{width}}, r \geq \frac{W_{height}}{C_{height}} \\ \left\lceil \frac{c - \frac{W_{width}}{C_{width}} + 1}{2} \right\rceil + \left\lceil \frac{r - \frac{W_{height}}{C_{height}} + 1}{2} \right\rceil \cdot \left(\frac{w - W_{width}}{2 \times C_{width}} + 1 \right) & c \geq \frac{W_{width}}{C_{width}}, r \geq \frac{W_{height}}{C_{height}} \end{cases} \quad (3\text{-}6)$$

$$WN_{hor} = \begin{cases} \left\lfloor \frac{c}{2} \right\rfloor + 1 & c < \frac{W_{width}}{C_{width}} \\ \frac{W_{width}}{2 \times C_{width}} & \frac{W_{width}}{C_{width}} \leq c < \frac{w - W_{width}}{C_{width}} \\ \left\lceil \frac{\frac{w}{C_{width}} - c}{2} \right\rceil & c \geq \frac{w - W_{width}}{C_{width}} \end{cases} \quad (3\text{-}7)$$

$$WN_{ver} = \begin{cases} \left\lfloor \frac{r}{2} \right\rfloor + 1 & r < \frac{W_{height}}{C_{height}} \\ \frac{W_{height}}{2 \times C_{height}} & \frac{W_{height}}{C_{height}} \leq r < \frac{h - W_{height}}{C_{height}} \\ \left\lceil \frac{\frac{h}{C_{height}} - r}{2} \right\rceil & r \geq \frac{h - W_{height}}{C_{height}} \end{cases} \quad (3\text{-}8)$$

For partial recognition, the cell position in each OSW is used to determine the corresponding MRToC values of the cell FV. Then, the partial squared Euclidean distances (PSEDs) between the cell FVs of each OSW and the corresponding portions of all reference vectors are calculated and accumulated for NNS classification of all cell-related OSWs. The obtained intermediate results are then stored in a PSED memory so that the next cell belonging to these OSWs can be processed for continuing the PSED calculation, as illustrated in Fig. 3-3.

SVM, which is widely employed in the literature, separates the positive and negative samples by a hyperplane with maximum-margin. Due to the limited relevance of different feature spaces, it becomes difficult to find an optimal hyperplane in the case of the simple combination of these features. In general, the pedestrian and the landscape often have different edge or texture feature information. Thus, the nearest-neighbor-search (NNS) classifier with hardware-friendly architecture produces different minimal-distance distributions for pedestrian and non-

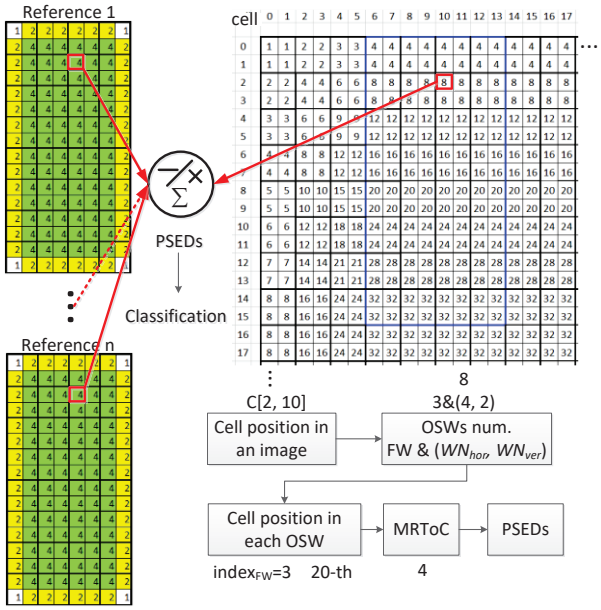


Fig. 3-5. Cell-based recognition for all OSWs to which the cell belongs.

pedestrian samples. It is proven that different feature spaces can comply with this distance-distribution manner.

3.2 Hardware Architecture

The developed five-stage pipelined architecture with dual-port histogram memories (DHMs) for cell-based HOG descriptor extraction and synchronization to the input pixels from the image sensor is depicted in Fig. 3-4. This architecture can be divided into three main functional parts, namely a control unit (CU), a pixel processing unit (PPU) for gradient-orientation (bin) and gradient magnitude calculation, and a vote unit (VU) with nine dual-port memories (DPMs) to store intermediate cell FVs. The pipeline registers synchronously latch input data with the same rising edge so that a computation result can be transferred to the following register in the pipeline. In other words, the proposed architecture consists of a pipeline processing cluster, comprising five stages pipelines.

3.2.1 Detector

For each pixel, the PPU firstly calculates its spatial derivatives G_x and G_y in horizontal and vertical directions, decodes the gradient orientation to bins, and computes the gradient magnitude *M(i, j)* according to (3-3). In the VU, *M(i, j)* is accumulated to the intermediate feature of the bin *k* that the current pixel belongs to. The VU contains nine magnitude storing units (MSUs) for accumulating the magnitudes appointed by the orientation bins. The MSU *k* in Fig. 3-4 includes a DPM *k* for storing the accumulated magnitudes for the bin *k*, a multiplexer MUXc to select the appropriate write-data for DPM *k*, and a multiplexer MUXd for write-

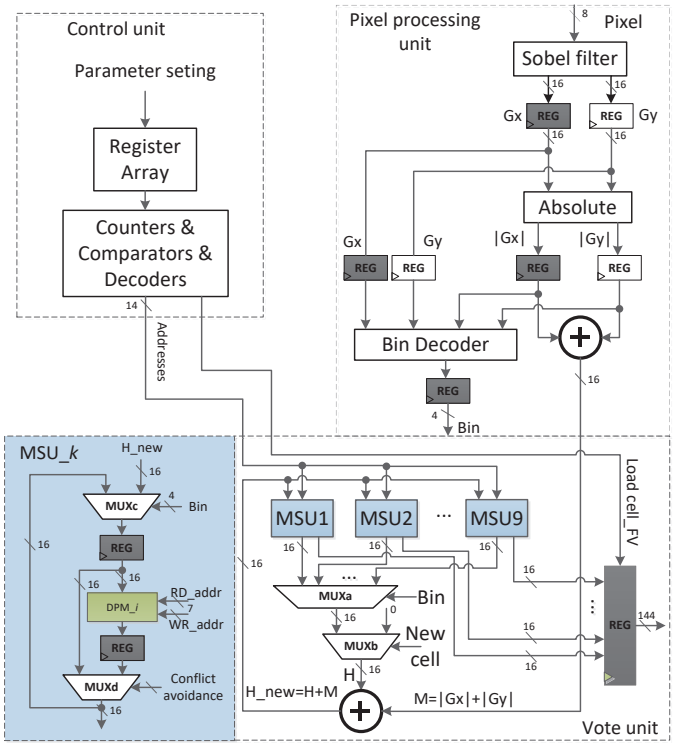


Fig. 3-7. Architecture for HOG feature-extraction. The whole architecture consists of three parts: control unit, pixel processing unit and vote unit.

conflicts avoidance. The multiplexer "MUXb" in the VU initializes the accumulated magnitude for each bin. When each bin is first selected for a processed cell, the "New cell (NC)" signal asserts to select "0". The VU outputs nine-dimensional cell-FVs when all the pixels within one cell have been processed. These cell-FVs are transmitted to the recognition part, where they are compared to the corresponding partial reference-vectors.

In this research, the angle range (-90°, 90°) is selected instead of the original range (0°, 180°) in (3-1) to avoid the singular point 90° of the tangent function. In the chosen angle range, the function $tan\theta$ is a monotonically increasing odd function with mirror symmetry. The nine evenly spaced bins over the range (-90°, 90°) are (-90°, -70°], (-70°, -50°], (-50°, -30°], (-30°, -10°], (-10°, 10°], (10°, 30°], (30°, 50°], (50°, 70°], and (70°, 90°). Due to the mirror symmetry, we only need to initialize four thresholds for bin assignment ($tan10°$, $tan30°$, $tan50°$, $tan70°$) rather than nine thresholds in the original implementation [38] since the other half of the thresholds ($tan(-10°)$, $tan(-30°)$, $tan(-50°)$, $tan(-70°)$) can be easily obtained through sign changing of the initialized ones. This also eliminates four multipliers and logic comparators in the hardware implementation.

Then, I fractionally store the nine-dimensional vectors of each cell in nine DPMs. Only the FVs of $w/8$ cells i.e. one cell row, have to be stored in the DPMs. The cell-feature storing-locations can be overwritten after the feature construction of one cell is completed. As a result, there is no height limitation of the image size due to the applied overwriting scheme of the DPMs. For example, a chip with 128×9 word DPMs can processed a maximum image width w=1024 pixels.

The described architecture can synchronize with the image sensor. Since the PPU processes one pixel in each clock cycle, the DPMs need to read and write at the same clock cycle. This concurrent reading and writing raise the risk of memory-access conflicts, caused by simultaneous accesses to identical memory addresses when adjacent pixels are assigned to the same bin. Actually, this situation is very common since the characteristics of adjacent pixels are with high probability similar except for the edge regions. The multiplexer MUXd is mainly utilized for avoiding this memory-conflict case in the MSUs. When adjacent pixels are assigned to the same bin, "k^{th}-d cell-feature" ($k \in [1, 9]$), writing is not necessary and avoided by selecting the previous accumulation result through signal "Conflict avoidance" as the correct operator for continued accumulation by addition of the current-pixel magnitude. In the meantime, the read operation of DPM k pauses until there is no further collision between read and write addresses. Otherwise, the output of DPM k is selected to update the magnitude-accumulation result for this bin. This conflict-avoidance method also guarantees continuity of the whole pipeline processing. In contrast to the frame-buffer-based and integral-image-buffer-based methods, the working frequency of the proposed architecture can synchronize to the image sensor at a relatively low working frequency for real-time processing with low power dissipation.

3.2.2 Gradient Generator and Vote

3.2.2.1 Sobel Filter for Gradient Calculation

The computation of gradient value and gradient direction must have low complexity and fit hardware-based implementations. The Sobel operator is used for calculating the gradient-values in x and y directions by convolution with a separable 3×3 Sobel mask and yields the spatial derivatives G_x and G_y for each pixel of the raw image. The gradient magnitude $M(i,j)$ for each pixel is then simplified according to Eq. (3-3) to avoid the root operation.

For each incoming pixel, two 2-dimensional convolution kernels (Fig. 3-5) are exploited for gradient calculation, working on the grayscale intensities of pixel-neighbors in a 3×3 pixel region. The incoming pixel (e.g., P5) is at the center of the mask and the line buffers (shift0,

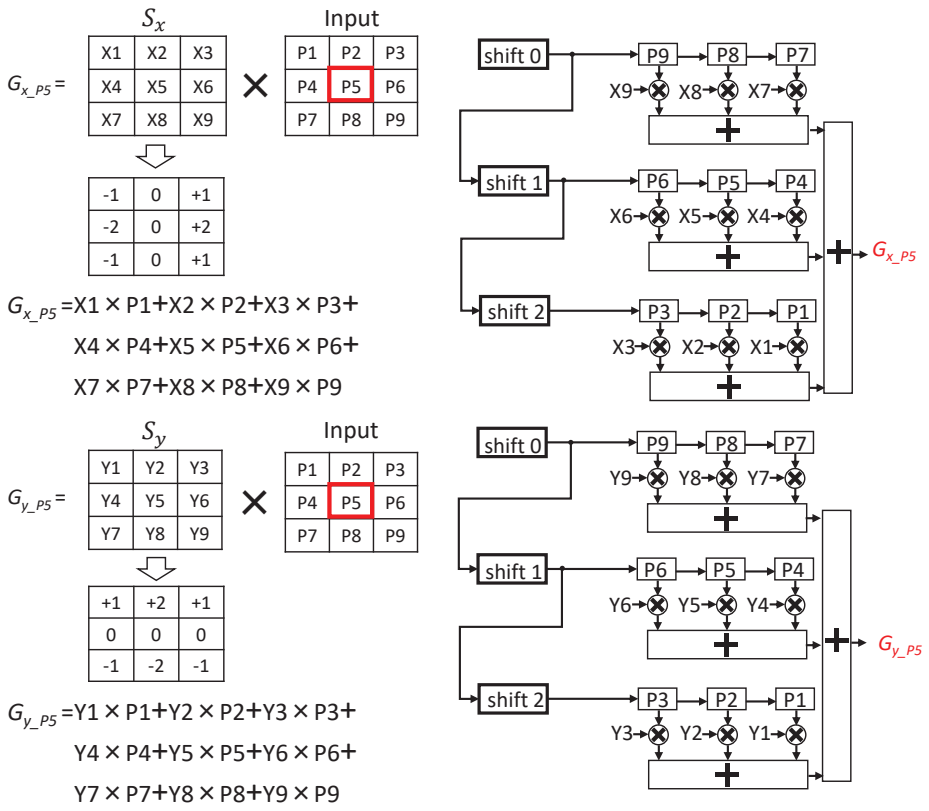


Fig. 3-9. Gradient calculations based on a Sobel filter with 3×3 kernel and an example for the pixel P5.

shift1, and shift2) produce the neighboring pixels in adjacent rows and columns. After line buffering, multipliers and adders calculate the convolution result for each central pixel. The two filters for G_x and G_y work in parallel and from their output data $M(i,j)$ can be obtained in the calculation unit. In the vote unit, $M(i,j)$ will be accumulated to corresponding histogram in the DPMs.

3.2.2.2 Bin Decoder for Orientation Calculation

The calculation of the arc-tangent function is computationally expensive, especially for hardware implementation. Most of the hardware friendly approximation algorithms adopt an iterative architecture resulting in deceleration of the system, or a lookup-table (LUT) method requiring large amounts of memory. Reference [42] proposed to combine the orientation calculation with the bin assignment by directly discretizing the pixel into bins according to its value of G_x and G_y, instead of computing the angle explicitly. The bin definition has been analyzed in Ref. [22] to demonstrate the influence of the bin number on performance. The results indicate that performance is improved significantly, but decreases with the number of bins, and when bin number is up to about nine, the difference becomes very small.

We optimized the approach further with respect to two aspects. First of all, we selected a more reasonable angle-range for the orientation to simplify the bin assignment. The function tanθ has an infinite set of singular points, e.g., 90°. This problem is fundamental to graphical properties of the tangent function, and the difficulty of bin decoding is increased if the singular point appears in the middle of the angle interval. In this paper, the angle range $(-90°, 90°)$ is selected instead of the original range $(0°, 180°)$ in eq. (1) to avoid this problem. In the selected angle-range, the function tanθ is a monotonically-increasing odd function with mirror symmetry. The nine evenly spaced bins over the range $(-90°, 90°)$ are shown in Fig. 3-6. Secondly, the value of |tanθ| is exploited in a look-up-table, so that only four tangent angles (bins) need to be initialized rather than nine bins, since the function |tan (θ)| is an even function in the selected domain, for example, $|\tan(-70°)| = |\tan(70°)|$. Finally, only half of the interval $(0°, 90°)$ has to be computed while the other half can be obtained by comparing the signs of G_x and G_y. Accordingly, the proposed method can eliminate five 23-bit signed multipliers in comparison to the solution of Ref. [44]. Finally, the angle computation is replaced by simple integer multiplications and logical comparisons as depicted in Eqs. (3-9) and (3-10) to determine the bin assignment (angel range).

$$\tan\theta_1 < \frac{G_y}{G_x} < \tan\theta_2 \tag{3-9}$$

$$G_x \cdot \tan\theta_1 < G_y < G_x \cdot \tan\theta_2 \tag{3-10}$$

The developed bin decoder is illustrated in Fig. 3-7 and calculates the orientation bins according to the computed horizontal and vertical gradients G_x and G_y. As shown in Fig. 3-8, three main cases i.e., $G_y = 0$, $G_x = 0$, and $G_y, G_x \neq 0$, can be summarized to arbitrate the final bin. At the beginning, four pairs of the $G_x \cdot \tan\theta$ are computed in parallel. To further simplify the calculation, the signs of G_x and G_y separate the first quadrant and the fourth quadrant. A fixed-point-number operation is adopted by left shifting the decimal tangent number, for

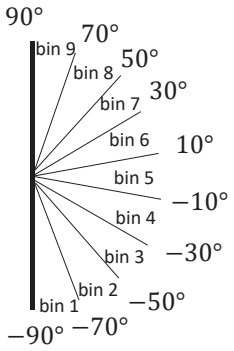


Fig. 3-11. Angular quantization into nine orientation bins for the range (-90°, 90°).

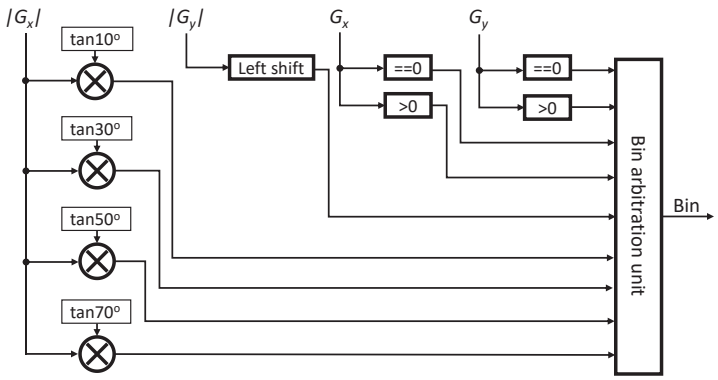


Fig. 3-13. Bin decoder with four multipliers and a bin arbitration unit.

example left shift 10-bit. At the same time, $|G_y|$ has to be shifted as shown in Fig. 3-7. Finally, the bin arbitration unit is a 9 × 9 channel matrix to generate the assigned bin based on G_x, G_y, $|G_x|$, and $|G_y|$ as illustrated in Fig. 3-8.

3.2.2.3 Parallelized Voting

The histogram vote circuitry contains nine parallelized voting elements (PVEs) for accumulating the vote values appointed by the orientation bins. Each PVE, as shown in Fig.3-9, includes a DHM unit for storing the accumulated-magnitudes of the vote value for nine bins and two multiplexes (MUXa and MUXb) for selecting the corresponding vote value according to the bin. When two adjacent pixels are assigned to the same bin, there is not enough time to write the updated vote value back to the DHM, and then to read it again for magnitude accumulation of the latter pixel. In this case the "Collision avoidance" signal in Fig. 3-9 thus selects the latched vote value after MUXb instead of reading it from the DHM. This also ensures the continuous pipeline processing for the histogram vote procedure. The comment multiplexer "MUXd" for the histogram vote circuitry can be used to initialize the accumulated-magnitude vote for every bin. When the process for a new cell begins, "New cell_row" signal

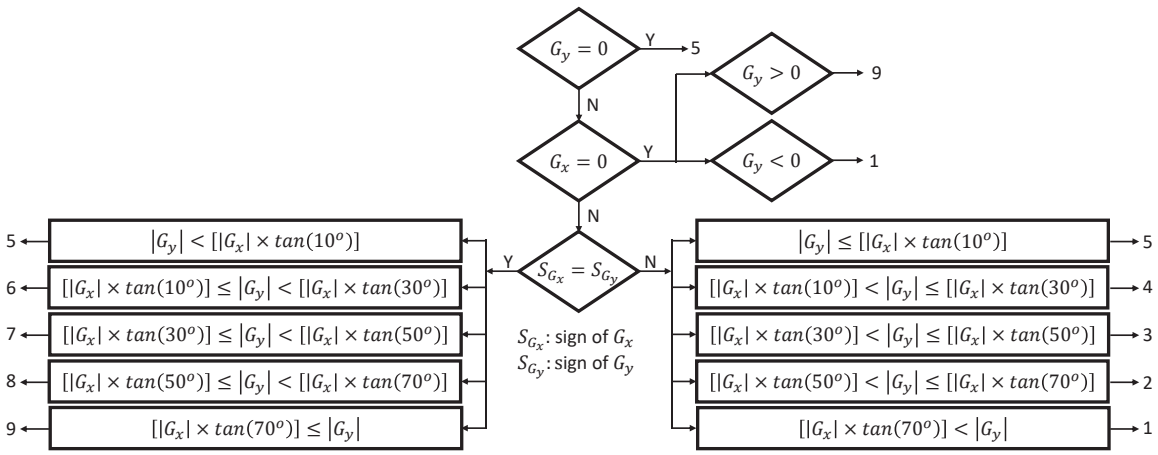


Fig. 3-15. Bin arbitration unit for calculating the final bin assignment.

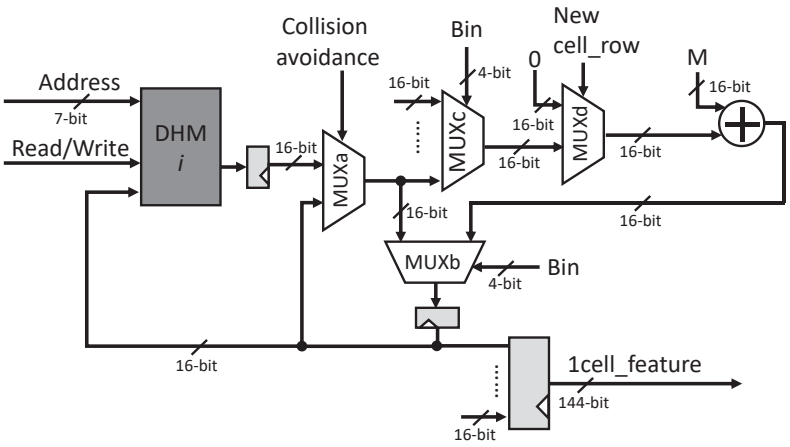


Fig. 3-17. Hardware architecture for the parallelized voting element.

asserts to select "0" as the accumulated vote value. As a result, only the magnitude of the first cell pixel will be written to the DPM.

In summary, the pixel contribution for the cell-based descriptor can be accumulated immediately after each pixel is transferred, so that the processing can be synchronized with image-sensor transmission. After the scanned line of the image sensor reaches to the height of a cell, the completed feature for one cell "1 cell feature" is transmitted to the parallel recognition circuit. For example, an x×y-pixel input image only requires a memory with x/8 words for storing one row of the cells in horizontal direction. In particular, each word of the dual-port memory has nine-bin× B (bit precision for each bin) bits, e.g., 144 bits when B is 16-bit. In other words, the width of the input image that this circuit can process is only determined by the memory size. On the other hand, there is no limitation of the image height since the DPM will be overwritten when the pixels for a new row of the cells are transferred from the image sensor.

3.2.3 Parallelized Cell-Based Recognition

Most conventional hardware designs would not take the feature extraction immediately when the pixels are inputted until the whole image is scanned due to the calculation for an integral image. This means they have an image-frame latency and are not real-time processing. In this paper, since the detection window is shifted in block units across the image in the sliding-window paradigm, the cell-based HOG descriptor is reused in multiple detection windows. Consequently, the cell-based descriptors are only computed once and reused according to their position in the image.

In general, the image width x should be larger than the detection window width with w pixels. The pixels are inputted in raster scan with line-by-line manner. There are (x-64)/16+1 detection

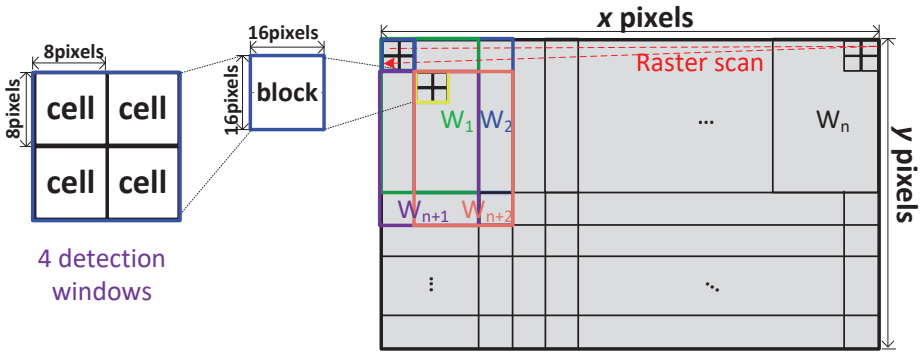


Fig. 3-19. A cell is normally located in several detection windows. And, the number of windows for a cell can be estimated by its position in the image.

windows in horizontal image direction in the case of 64×128-pixel windows, 16×16-pixel blocks, and 8×8-pixel cells. Furthermore, the feature vector of a detection window can be completed only after the currently input image line reaches the last line of the window, i.e., the 128 image line from the top of the window.

As shown in Fig. 3-10, a cell is usually located in several search windows. The reuse time of a cell can be estimated by the cell location in an image. And then, the number of windows for this cell C[c,r] can be calculated from its position in column c and row r of the image. Given an input image with x×y pixels, the total number of the detection windows can be summarized and initialized in a look-up table.

Finally, the nearest-neighbor-search (NNS) classifier is applied to recognize the objects in each window according to the Euclidean distance between the HOG descriptor of each detection window and the reference vectors, as illustrated in Fig. 3-11. The 3780-dimensional HOG feature vectors are categorized into three groups (corner, edge, internal) to determine the reuse time of the current cell C[c,r]. Generally, each of the simultaneously processed windows is in a different stage of its feature-vector construction and the partial squared Euclidean distance

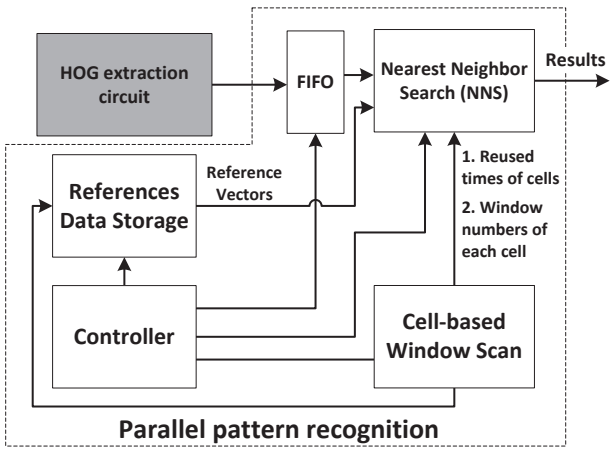


Fig. 3-21. Block diagram of the hardware architecture for parallel pattern recognition.

calculation with respect to the reference feature-vectors.

3.3 Implementation and Results

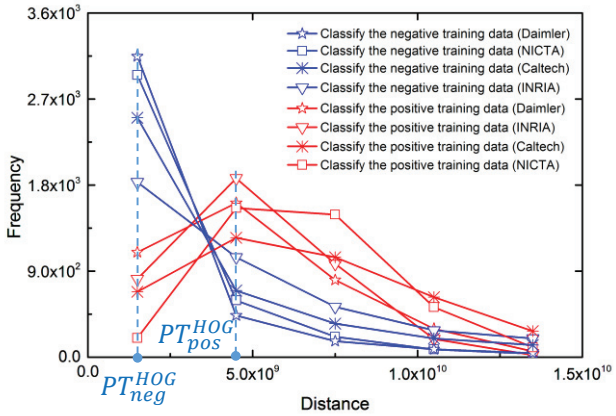
3.3.1 Detection Accuracy

Simulation results by software implementation are used to evaluate the effects of the cell-based FVs, the sliding-window recognition, and the complementary dual-feature space on the accuracy performance of the complementary nearest neighbor classifier (CNNC). We confirmed the impact of the proposed algorithms on human-detection accuracy using the INRIA and NICTA [45] human dataset and two evaluation criteria: true positive rate per window (TPPW) and true negative rate per window (TNPW). TPPW and TNPW measure the performance of correct classification for test windows which contain humans and which do not contain humans, respectively.

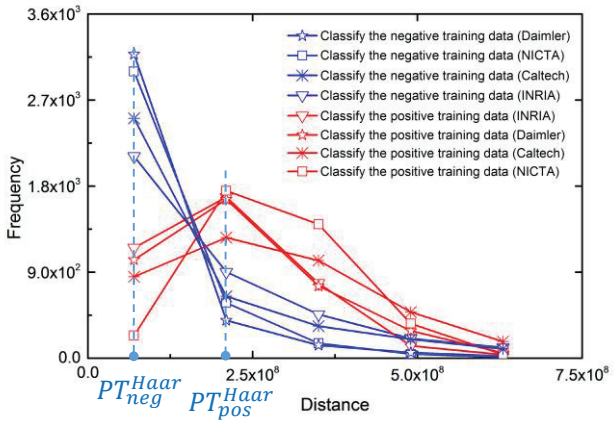
The INRIA human dataset, providing a more challenging human detection problem [22], was used as the base-data set for comparison in many human-detection investigations. The human images in the dataset are challenging due to the various lighting conditions, complex postures, partial occlusion, and complex backgrounds. The data set contains 2416 positive training images and 1218 large images, which contain no humans. From these 1218 large images, 12180 negative training images can be extracted. For the classification test, we used different sets of 1126 positive testing images and 453 large images which contain no humans. From these large images, 4530 negative testing images were extracted.

The NICTA data set is a large-scale urban dataset with a significant number of pedestrians. The training dataset contains 37339 positive images in which there are 25551 unique pedestrians and 200000 negative images. The test data set has 6877 positive images and 50000 negative images.

First of all, the cell-based HOG and Haar-like FVs, extracted from the training images, were clustered by the k-means algorithm. Then, the resulting centroids of the clustering process were taken as the reference FVs for classifying the training dataset to calculate the PT_{pos}^{HOG}, PT_{pos}^{Haar}, PT_{neg}^{HOG}, and PT_{neg}^{Haar} as shown in Fig. 3-12. Actually, a human has much more edge or texture characteristics than the scene background. As a result, the positive samples tend to have much larger classification distances than the negative samples. We can conclude that the distance histograms for non-normalized HOG and Haar-like feature have the same tendency in different datasets (INRIA, NICTA, Caltech Pedestrian Detection Benchmark ([46]-[47]), and Daimler Mono Pedestrian Detection Benchmark Dataset [48]). Hence, the PT_{pos}^{HOG}, PT_{pos}^{Haar}, PT_{neg}^{HOG}, and PT_{neg}^{Haar} are robust to different datasets. Next, using the defined PTs to



(A) Distance histogram of the nearest neighbors with HOG feature



(B) Distance histogram of the nearest neighbors with Haar-like feature

Fig. 3-23. Definition of the priority threshold (PT) for HOG and Haar-like features derived from four different standard datasets. HOG and Haar-like features comply with the same distance-distribution manner in these datasets.

classify the corresponding test datasets and presenting the graphs of the obtained TPPW and TNPW performance, respectively, versus the number of cluster centroids (reference FVs) used in the k-means training phase.

Due to the cell-based sliding-window recognition, this work has applied no block-based normalization, which is considered as an important processing step in the original study of [22] and the hardware implementations of ([35]-[39], [49]).

As illustrated in Fig. 3-13, the truncation of the bit precision and non-normalization lead an average loss of 0.05% and 7.5% in TPPW and TNPW for INRIA dataset and an average loss of 1.5% and 8.2% for NICTA dataset with individual HOG feature. As for the Haar-like based detection, the cell-based feature extraction scheme is found to show better results in TPPW with an average improvement of 5% and a degradation in TNPW with an average loss of 4.6% for INRIA dataset. Meanwhile, the accuracy has 0.9% loss in TPPW but 3.5% improvement in TNPW with Haar-like feature for NICTA dataset.

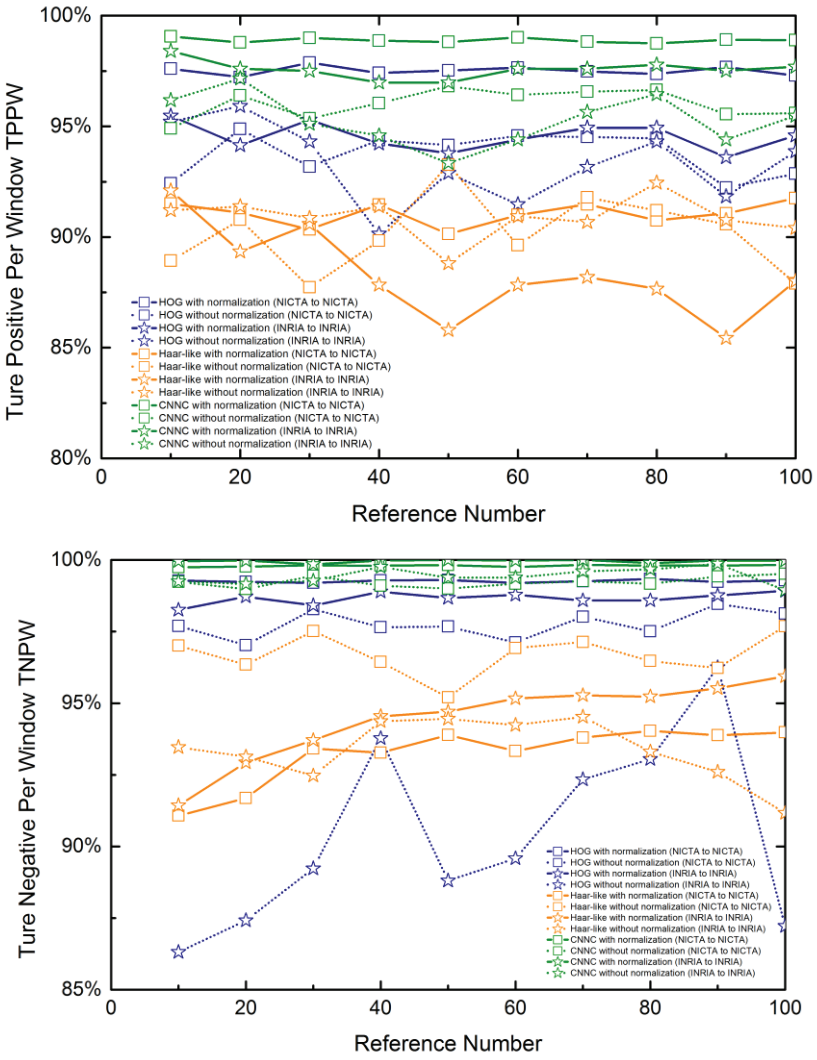


Fig. 3-25. Comparison of the TPPW and TNPW performances between the classification by the block-based algorithm with normalization (solid lines) and our cell-based algorithm without normalization (dashed lines) in two different datasets. Dual-feature classification with CNNC achieves the best TNPW results.

However, the classification performance of the proposed CNNC architecture with complementary dual-feature space is found to achieve an average of 95.3% in TPPW and 99.4% in TNPW for INRIA dataset. At the same time, NICTA dataset achieves an average accuracy of 96% in TPPW and 99.2% in TNPW. The CNNC significantly outperforms the individual HOG or Haar-like feature-based classification with and without normalization. Hence, the accuracy loss from the hardware implementation can be compensated by the CNNC in dual feature space. As well as, in contrast to the individual feature, the results of the CNNC are more stable for normalized and non-normalized features since the dual feature space can supplement each other during the classification stage. The ASIC implementations in [39] and [49] have also been compared to the original HOG+SVM framework in [22]. We implemented the

approach of [22] and obtained the results of 94.2% for TPPW and 99.5% for TNPW. In this paper, the cell-based feature extraction scheme without normalization leads a further reduction of the memory utilization and a straightforward parallelization of the sliding-window recognition. Furthermore, the complementary mechanism, which exploits different aspects of different feature spaces, consequently also produces better classification results with 2.2% TPPW and 0.3% TNPW improvements in comparison to the ASIC implementations of the HOG+SVM algorithm in ([35][36][39][49]). Comparing to our work, the accuracy loss in ([39][49]) is controlled by complying with the original framework in [22]. As a tradeoff, one of the requirement for more hardware resources is the normalization circuit in ([39]-[49]).

Because the main extraction domain of each feature emphasizes different image aspects, each feature's accuracy domain is different and accurate recognition is high for these emphasized special characteristics. As shown in Fig. 3-14, object (A) is difficult to be distinguished by HOG due to the unclear edges. On the other hand, the Haar-like feature captures more efficiently the texture properties. It can, therefore, be inferred that the consideration of dual features with complementary characteristics enlarges the overall feature accuracy domain and can thus improve the recognition accuracy efficiently. The feature extraction sub-component of the developed hardware architecture implements the feature extraction for both HOG and Haar-like descriptors. Theses dual features are used to complement each other in the CNNC efficiently. The HOG descriptor attempts to characterize objects by their distribution of local intensity gradients, while the Haar-like descriptor can better describe the detailed internal characteristic of an object.

The CNNC consists of a number of NNS classifiers in each of which the reference data is off-line trained by the unsupervised k-means clustering algorithm in an individual feature space,

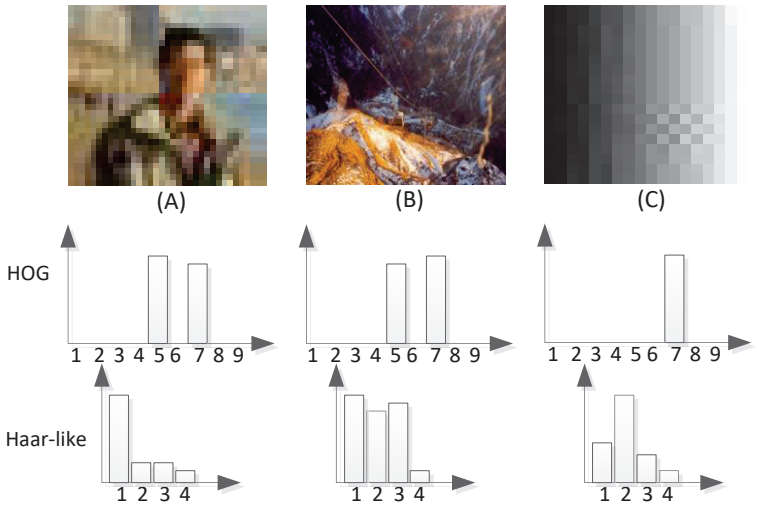


Fig. 3-27. Feature emphasis for describing different scenes.

in order to reduce the number of references for minimizing the computational efforts in the classification stage. The obtained prototypes are the centroids of clusters derived from the average of the clustered training samples. The number of prototypes, produced by the k-means clustering algorithm, is sufficiently compressed to a much lower order than that of the training samples.

In this research, we capture the human-figure edge characteristics by the histogram of oriented gradient (HOG) descriptor and the detailed internal human-figure characteristics by the Haar-like descriptor, both complementing each other to describe the human figure more completely. Because the main domain of each descriptor lays emphasis on different aspects, each descriptor's accuracy domain is different and recognition accuracy is high for these specific aspects. It can be inferred that different descriptors with complementary characteristics have their own accuracy domains and that their combination can improve the overall recognition accuracy substantially.

The basic idea of the CNNC is, therefore, to combine different perspectives of the multi-domain descriptors to improve the detection performance, even though the accuracy performance of each individual classifier may under-perform state-of-the-art detection systems as e.g. SVM. For the classification mechanism, instead of simply combining dual features, CNNC emphasizes the classification result of the nearest-neighbor-search classifier in each feature space so that results can complement each other through the different perspective of each classifier.

In fact, a human figure often has different edge or texture characteristics from the scene view. One behavior of the NNS classifier is that the positive samples tend to have more noticeably various classification distances than the negative samples. Accordingly, a distance for HOG or Haar-like feature space, which is closer to the largest distribution frequency of the positive or negative samples, is motivated to provide higher confidence when the classification results of the dual features are different.

A priority thresholds (PT) for each feature space helps to choose the classification as the result of the CNNC when the individual classifier for different feature outputs different classification. The final classification result emphasizes the feature space with higher confidence which is expressed by the relative distance of the classification distance in each feature space to the PT. For each feature space, the PT can be determined in the training stage as following:

Step 1: Obtain k prototypes (reference data) by independently clustering the positive and negative training dataset with k-means clustering algorithm.

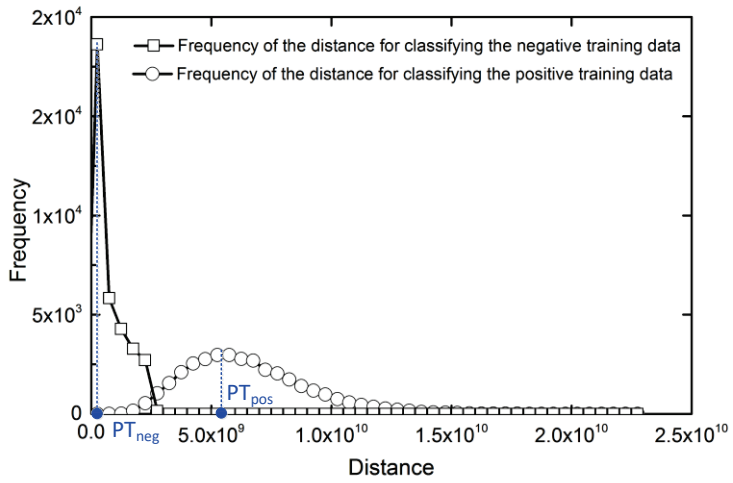


Fig. 3-29. Distance histogram of the nearest neighbors using the prototypes classifying the entire training dataset with non-normalized features for training the PT.

Step 2: Classify the entire training dataset with the 2k prototypes and plot the distances of nearest neighbors for the correct classifications in Fig. 3-15.

Step 3: Calculate the distance histograms for positive and negative training dataset, respectively.

Step 4: Choose the distance intervals of the histogram with largest counting numbers (peaks) for positive and negative histogram respectively and calculate the average values of the distance intervals as priority thresholds (PT_{pos} and PT_{neg}). According to the experimental results for different datasets, PT_{pos} is always much larger than PT_{neg} while we adopt non-normalized features.

Then, determine the confidence parameters (CP) that show the distance relationships for the PT_{pos} and PT_{neg} to the nearest-neighbor distance of the input vector to the $2k$ prototypes (D_{nns}) according to (3-11) and (3-12).

$$CP_{pos} = \begin{cases} \frac{D_{nns}-PT_{pos}}{D_{nns}-PT_{neg}} & D_{nns} > PT_{pos} \\ \frac{|D_{nns}-PT_{pos}|}{PT_{pos}-PT_{neg}} & PT_{neg} < D_{nns} < PT_{pos} \\ \frac{|D_{nns}-PT_{pos}|}{PT_{pos}-D_{nns}} & D_{nns} < PT_{neg} \end{cases} \quad (3\text{-}11)$$

$$CP_{neg} = \begin{cases} \frac{D_{nns}-PT_{neg}}{D_{nns}-PT_{neg}} & D_{nns} > PT_{pos} \\ \frac{D_{nns}-PT_{neg}}{PT_{pos}-PT_{neg}} & PT_{neg} < D_{nns} < PT_{pos} \\ \frac{|D_{nns}-PT_{neg}|}{PT_{pos}-D_{nns}} & D_{nns} < PT_{neg} \end{cases} \quad (3\text{-}12)$$

Finally, the classification mechanism for CNNC with two feature descriptors (HOG and Haar-like) in (3-13) chooses the classification result of the feature space with high *CP* that has smaller relative distance of the D_{nns} and the PT.

The PSED for every OSW is invoked from the memory for the final distance comparison. Meanwhile, the cell position in each OSW determines the components of the reference FVs corresponding to the extracted cell FV. Above operation is repeated for the processing the $WN_{hor} \times WN_{ver}$ OSWs, which contain this cell.

$$C_{CNNC} = \begin{cases} 1 & C_{Hog}=1,\ C_{Haar}=1 \\ -1 & C_{Hog}=-1,\ C_{Haar}=-1 \\ 1 & C_{Hog}=1,\ C_{Haar}=-1 (CP_{pos}^{HOG} > CP_{neg}^{Haar}) \\ -1 & C_{Hog}=1,\ C_{Haar}=-1\ (CP_{pos}^{HOG} < CP_{neg}^{Haar}) \\ 1 & C_{Hog}=-1\ ,\ C_{Haar}=1 (CP_{neg}^{HOG} < CP_{pos}^{Haar}) \\ -1 & C_{Hog}=-1,\ C_{Haar}=1\ (CP_{neg}^{HOG} > CP_{pos}^{Haar}) \end{cases} \quad (3\text{-}13)$$

As illustrated in Fig. 3-16, synchronized with the pixel-based HOG and Haar-like feature extraction, the pixel coordinates are also converted to the position of the processed cell in the image frame for simultaneous calculation of the corresponding *FW*, WN_{hor}, and WN_{ver} data.

At the beginning of the recognition procedure, the index of the OSW is set to *FW*. Then the index increases one by one from *FW* to reach $FW+WN_{hor}$ in the horizontal direction. After processing each row of the OSW matrix, the index of the OSW is re-assigned to FW+*xn*, where $n = (w\text{-}W_{width})/2 \times C_{width}$ and *x* ($0 \leq x < WN_{ver}$) are the maximum OSW number in the horizontal direction of the input image with $w \times h$ pixels and the row number of the OSW matrix,

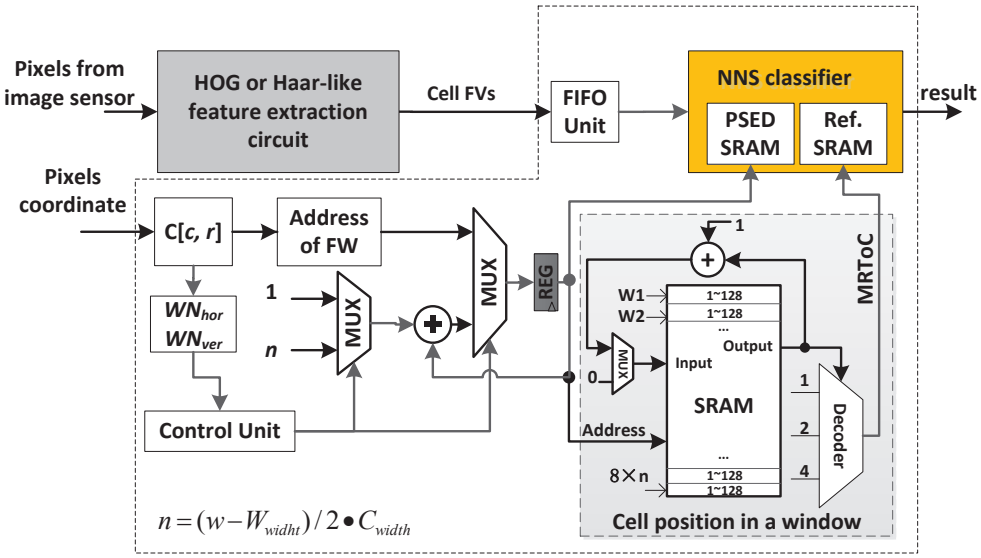


Fig. 3-31. Block diagram of the hardware architecture for parallel cell-based recognition. The number of OSWs can be deduced from the cell position in an image. For each OSW, the cell position in the window determines the MRToC value.

respectively. The recognition processing terminates when the index of the OSW reaches to $FW+(WN_{ver}-1)\times n+WN_{hor}$. A control unit with finite state machine manages the increasing operation for the index of the OSW. After each index increment, the cell position in the corresponding OSW is used to read out the MRToC value from the SRAM block, which can be reused after eight OSW-rows to achieve a good memory efficiency. The feedback loop in lower right of the Fig. 3-16 is used to increase the cell position in an OSW.

The offline trained references for HOG and Haar-like features are independently initialized in the "Ref. SRAM". The NNS circuitry in Fig. 3-17 with a partial storage concept and a parallel-pipelined computation architecture ([50]-[53]) is applied for classification. In other words, each reference memory stores a number of partial references. The parallelism, which is equivalent to the number of the parallel reference memories, is different depending on the dimensionality of the cell FV. In particular, in the same way, as for the storage concept of the MRToC SRAM, the PSED storage unit only has to store $8 \times n$ intermediate results. Finally, the label of the reference vector with minimal SED is outputted as the recognition result for each OSW.

As illustrated in Fig. 3-14, the recognition processing can already be started from the $(7w+7)$-th pixel of an image frame and has to be completed with respect to the contribution from the 1st cell row before the first cell FV of the 2nd cell row (at $(15w+7)$-th pixel) is determined. In

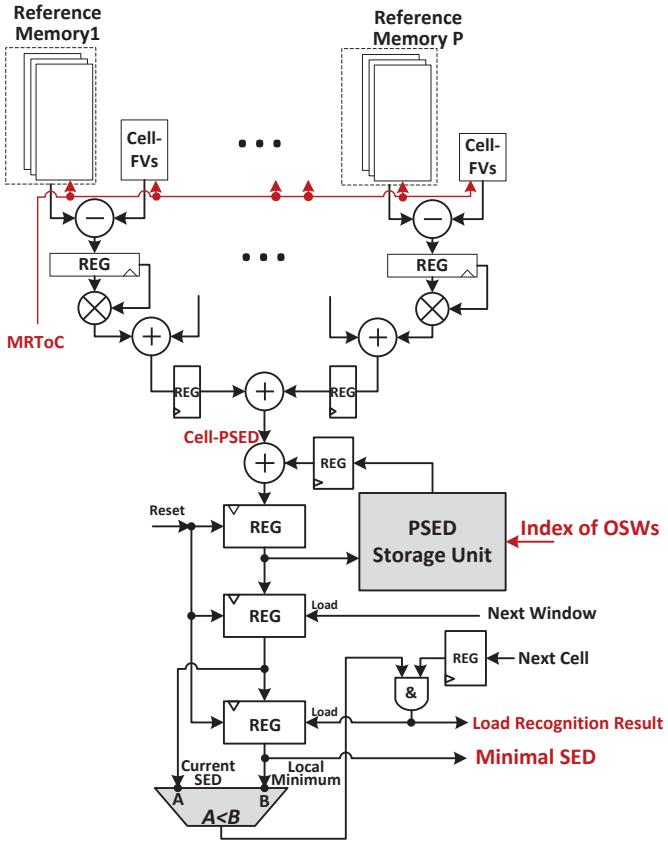


Fig. 3-33. NNS circuits for individual HOG and Haar-like descriptors.

other words, the recognition-process contribution from one cell row only has 8w clock cycle to finish. Otherwise, the FIFO-length between the feature-extraction unit and the recognition circuitry must be increased so that it is large enough to cover the mismatch between the two system parts. This could amount to the necessity of having to buffer the cell FVs of almost an entire image frame in this FIFO. To resolve this constraint, we use two different clock frequency domains in the feature extraction (f_{FV}) and recognition part (f_{RE}) of our system architecture, where the recognition circuitry has a higher working frequency to finish the required processing fast enough, i.e. $f_{RE}=4f_{FV}$ in this work. With above requirements fulfilled, the latency T for the application of pedestrian detection in an image frame becomes only $\frac{(w\times h)+8w}{f_{FV}}$ ms where f_{FV} is the working frequency of the image sensor, to which the whole system architecture is synchronized. In case of a XGA size camera STC-MC83PCL (1024×768 pixels), T is about 26.94 ms while the pixel frequency (f_{FV}) is 29.5 MHz at 29.18 fps. In the meanwhile, the working frequency of the recognition unit (f_{RE}) is 118 MHz.

3.3.2 Post-Layout Results

A test chip (see Fig. 3-18) was fabricated in 180 nm CMOS technology to implement the architecture for HOG descriptor extraction by the cell-based scan method with synchronization to the pixel input. Total chip area is 1.59 mm^2 where the on-chip dual-port memory of 2.25 KB for the nine 128×16-bit DHMs consumes about 70% area. The word precision for G_x, G_y, and the histogram values is 16 bit, which provides reasonable classification accuracy and minimizes hardware cost. Power consumption is 42.3 mW at measured maximum frequency of 120 MHz at 1.8 V supply voltage.

The chosen DPM configuration for storing the x/8 intermediate HOG descriptors of an image row allows to handle a maximum input-image width of 1024 pixels, while the input-image

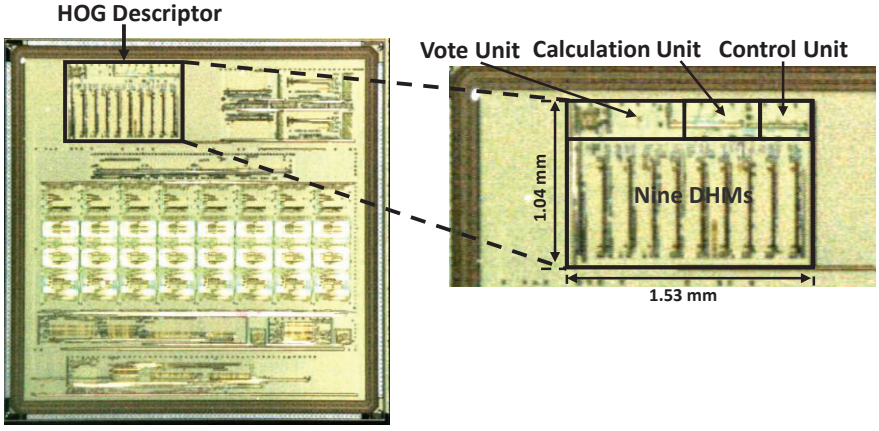


Fig. 3-35. Micrograph of the fabricated chip in 180 nm CMOS technology.

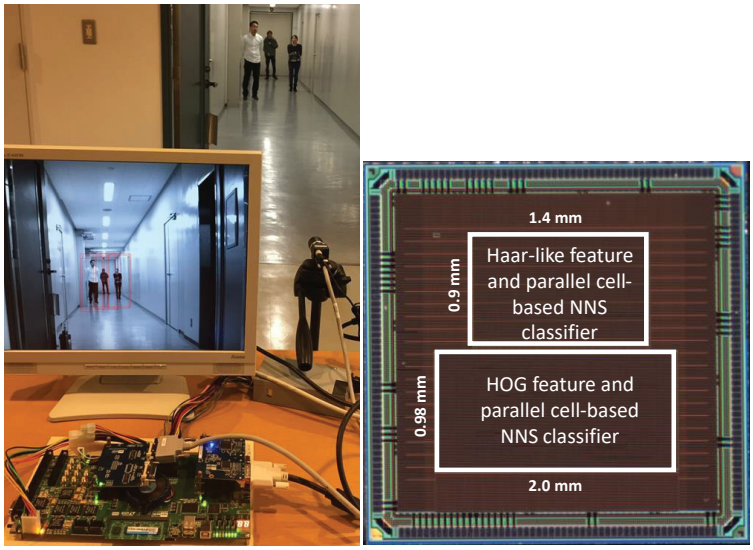


Fig. 3-37. Micrograph of the prototype chip in 65 nm SOTB CMOS technology and the FPGA-base demonstration system with XGA camera and single-scale sliding window. HOG descriptor and Haar-like descriptor are integrated with a dedicated cell-based NNS classifier, respectively.

height is unlimited. For the application example of XGA (1024 × 768) resolution videos, HOG-feature vectors can be extracted at 120 MHz operating frequency with a maximum frame rate of 122 fps.

As an update of the 180 nm porotype, a test chip fabricated in 65 nm SOTB CMOS technology, which is depicted in Fig. 3-19, verifies the CNNC described above. The test chip implements dedicated feature extraction and parallel OSW recognition with NNS circuits for HOG descriptor and Haar-like descriptor, respectively. Due to the cell-based sliding window recognition, and an overwriting scheme for obsolete intermediate data, the prototype chip achieves high area-density and memory-utilization efficiency with a core area of 3.22 mm^2, a memory consumption of 0.602 Mbit, and an average power consumption of 75.48 mW at 200 MHz and 1 V.

The recognition circuits based on the HOG feature embeds 394 Kbit SRAM, in which 18 Kbit SRAM is used for cell-feature construction, 176 Kbit SRAM is reference-data memory, 128 Kbit SRAM is for PSED intermediate storage, and 72 Kbit SRAM serves as FIFO for cell-FV buffering. As for the Haar-like part, the overall SRAM consumption is 208 Kbit, including 64 Kbit SRAM for cell-feature construction, 96 Kbit SRAM for reference-data storage, 16 Kbit SRAM for PSED intermediate storage, and 32 Kbit SRAM as FIFO for cell-feature buffering. The word precision for the histogram values G_x, G_y and the Haar features D_x, D_y is 16 bit, in order to achieve reasonable classification accuracy and minimization of hardware cost.

Table 3-I. PERFORMANCE COMPARISON TO PREVIOUS WORK.

	[39]	[49]		Our work
		Column scan	Row scan	
CMOS technology	65 nm	45 nm SOI		65 nm SOTB (SOI with thin gate oxide and BOX layers) CMOS
Feature descriptor	HOG	HOG		HOG & Haar
Feature-core number	Dual HOG cores	Triple HOG cores		Dual complementary cores: single HOG core & single Haar-like core
Classifier	SVM-based	SVM-based		NNS-based
Power dissipation (mW)	99.52 (42.9 MHz at 1.1 V)	45.3 (270 MHz at 0.72 V)	58.5 (270 MHz at 0.72 V)	75.48 (200 MHz at 1 V, 44.96 for HOG & NNS, 30.52 for Haar-like & NNS)
Storage size (M-bit)	1.22	0.538	1.121	0.602 (0.394 for HOG & NNS, 0.208 for Haar-like & NNS)
Core size (mm^2)	3.96	2.688	3.456	3.22 (1.96 for HOG & NNS, 1.26 for Haar-like & NNS)
Image resolution	FHD (1920 × 1080 pixels) at 30 fps at 110 MHz	FHD (1920 × 1080 pixels) at 60 fps at 270 MHz		1024 × ∞ pixels E.g., (1024 × 1616 pixels) at 30 fps at f_{FV}=50MHz and f_{RE} =200 MHz
Flexibility for image size (pixels)	only 1920×1080	only 1920×1080		≤1024 × ∞
Energy consumption*	1600 pJ/pixel	364 pJ/pixel	470 pJ/pixel	906 pJ/pixel for HOG based recognition, 615 pJ/pixel for Haar-like based recognition

*Energy consumption = power dissipation/(image resolution×fps).

3.3.3 Architecture and Algorithmic Optimization Results

A performance comparison between our work, which exploits dual complementary feature space and other ASIC implementations using the HOG + SVM framework [39], [49]is illustrated in Table 3-I. Both [39] and [49] can process single-scale FHD (1920 × 1080 pixels) videos. Our design is able to handle a maximum video size of 1024 × 1616 pixels in row scan manner. The memory for storing the w/8 intermediate cell FVs, implemented in the prototype design, allows dealing with a maximum input-image width of 1024 pixels, while the input-image height is only limited by the processing speed requirements (e.g. fps). Maximum input-image width can be easily increased in our architecture by using a larger memory size for intermediate cell-FV storage.

In [39], dual HOG cores are employed to process single-scale images. In [49], the design supports multi-scale detection for fixed image resolution. Instead of limitation to a fixed image

resolution, the reported prototype for the proposed architecture realizes a resolution flexibility of up to $1024 \times \infty$ pixels, where only the actually implemented memory capacity for intermediate cell FVs limits the maximum image width. Note that the image-resolution flexibility of the proposed architecture can be exploited to support multi-scale processing. Our work with multiple chips can allow enlarging or reducing the scale of target objects in images to match the sizes of the detection window and to-be-recognized target objects. Even though our chip is flexible to process images with different image resolution, it consumes less memory because of synchronization between pixel-data transmission and clock frequency for processing, overwriting of obsolete data in the cell FVs storage memory, and progressive cell-based partial recognition as soon as cell FVs become available.

To demonstrate the robustness of the PT for different dataset, we classify the INRIA test dataset by the trained reference data from the NICTA training data (NICTA to INRIA). Then, using the references from the INRIA training data recognize the NICTA test data (INRIA to NICTA).

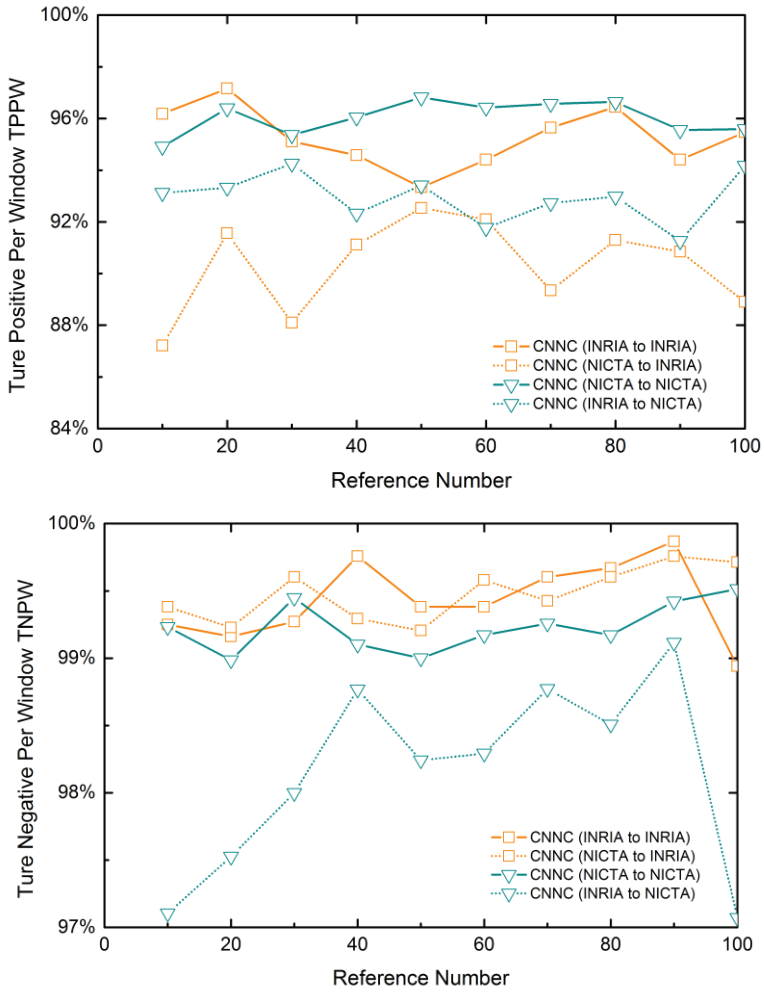


Fig. 3-39. Comparison of the TPPW and TNPW performances using the references from different training datasets to classify different test datasets.

The results in Fig. 3-20 prove that CNNC is still efficient in the cross verification. In case of NICTA to INRIA, the average accuracy of the CNNC is 90.3% in TPPW and 99.5% in TNPW. With respect to INRIA to NICTA, TPPW rate is 92.9% and TNPW rate 98.1%. The accuracy loss shows that the compatibility of a dataset is limited due to different camera sensors and illumination conditions.

In addition, the scale of the training data has small effects on the accuracy performance according to Fig. 3-20. However, the proportion of the positive and negative samples can affect the TPPW and TNPW. Even though the scale of the NICTA dataset is much larger than that of the INRIA dataset, only the proportion of the positive and negative samples ($P = \frac{Num_{pos}}{Num_{neg}}$) affects the accuracy performance. In the case of the INRIA, $P_{INRIA} = 0.198$. For NICTA, $P_{INRIA} = 0.187$. The more positive sample can more effectively achieve higher TPPW. Furthermore, in Fig. 3-21, the number of negative training sample keeps the same while we choose 1k, 10k, 20k, and the entire positive training samples. We can prove that the

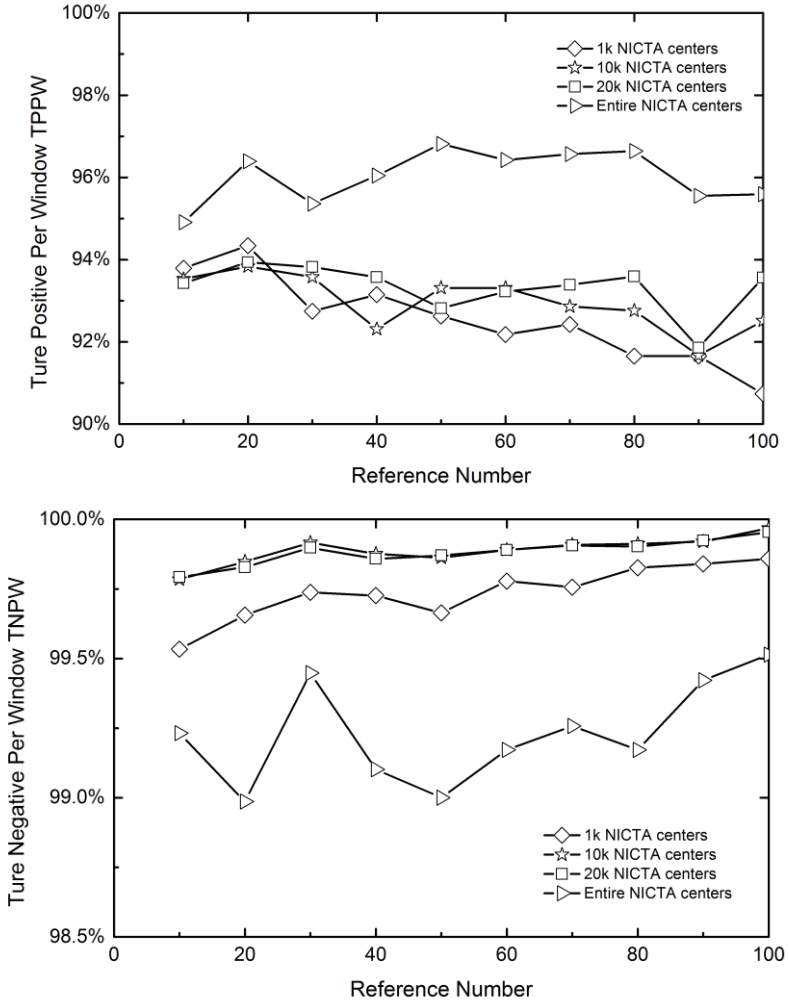


Fig. 3-41. Comparison of the TPPW and TNPW performances between different scales of the positive training samples.

proportion of positive and negative training samples rather than the scale affects the accuracy performance of the prototype-based nearest neighbor classifier.

3.4 Summary

The chapter introduces a hardware-oriented HOG algorithm which exploits the cell-based scan strategy. The design scheme enables image-sensor synchronization and extraction-speed acceleration. Furthermore, buffers for image frames or integral images are avoided. An image-size scalable hardware architecture with an effective bin-decoder and a parallelized voting element (PVE) is developed and used to verify the hardware-oriented HOG implementation with the application of human detection. The fabricated test chip in 180 nm CMOS technology achieves fast processing speed and large flexibility for different image resolutions with substantially reduced hardware cost and energy consumption. The sliding-windows shift across the image in steps of overlapped blocks so that each cell can belong to more than one block and more than one window, so that cell-feature vectors appear several times in a single window-feature vector and also in different window-feature vectors. This results in a large number of repeated calculations of cell histograms in the block-based algorithm, since most of the cells need to be recalculated for the feature construction of different blocks.

To further improve the accuracy, a complementary nearest neighbor classification (CNNC) architecture using HOG and Haar-like feature spaces, cell-based feature-vector extraction and parallel sliding-window classification is developed. A coprocessor prototype in 65nm SOTB CMOS for pedestrian detection has good energy and Si-area efficiency, high classification accuracy, and fast detection-speed performance. The embedded cell-based HOG and Haar-like descriptor extraction units apply a pixel-based pipelined architecture, can be synchronized to the working frequency of the image sensor, and do not need any image-frame or integral-image buffer memories and have the flexibility for processing different input-image sizes. The image-size flexibility also enables classification operation with multiple scaled images for detection of objects with variable sizes. The cell-based sliding-window mechanism leads to parallel classification capability for all overlapping windows that contain the currently processed cell.

References

[1] N. M. Botros, and M. Abdul-Aziz, IEEE Trans. Ind. Electron. 41, 665 (1994).

[2] D. C. Hendry, A. Duncan, and N. Lightowle, IEEE Trans. Neural Networks 14, 1085 (2003).

[3] C. Kyrkou, and T. Theocharides, IEEE Trans. Comput. 61, 831 (2012).

[4] L. Li, W. Huang, I. Y. H. Gu and Q. Tian, IEEE Trans. Image Process. 13, 1459 (2004).

[5] E. Shechtman and M. Irani, Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2007, p. 1.

[6] C. Li and K. M. Kitani, Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2013, p. 3570.

[7] Ø. D. Trier, A. K. Jain and T. Taxt, Pattern Recognition 29, 641 (1996).

[8] J. A. Kalomiros and J. Lygouras, Microprocess. Microsyst. 32, 95 (2008).

[9] A. L. Yuille, P.W. Hallinan and D. S. Cohen, Int. J. Comput. Vision 8, 99 (1992).

[10] L. Zhang, L. Zhang, D. Tao and X. Huang, IEEE Trans. Geosci. Remote Sens. 51, 242 (2013).

[11] F. An, T. Akazawa, S. Yamasaki, L. Chen, and H. J. Mattausch, Jpn. J. Appl. Phys. 54, 04DE05 (2015).

[12] X. Zhang, F. An, L. Chen and H. J. Mattausch, Jpn. J. Appl. Phys. 55, 04EF02 (2016).

[13] F. An, and H. J. Mattausch, J. Syst. Archit. 59, 155 (2013).

[14] F. An, X. Zhang, L. Chen and H. J. Mattausch, IEEE ISCAS, 2016, p. 1338.

[15] P. F. Felzenszwalb and D.P. Huttenlocher, Int. J. Comput. Vision 61, 55 (2005).

[16] S. Ioffe and D.A. Forsyth, Int. J. Comput. Vision 43, 45 (2001).

[17] K. Mikolajczyk, C. Schmid and A. Zisserman, Proc. IEEE European Conf. Computer Vision, 2004, p. 69.

[18] A. Tejani, D. Tang, R. Kouskouridas and T. K. Kim, European Conf. Computer Vision, 2014, p. 462.

[19] L. A. Jeni, J. F. Cohn and T. Kanade, 11th IEEE Int. Conf. Workshop Automatic Face and Gesture Recognition 2015, vol. 1, p. 1.

[20] P. Dollár, C. Wojek, B. Schiele and P. Perona, Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2009, p. 304.

[21] M. Enzweiler and D. M. Gavrila, IEEE Trans. Pattern Anal. Mach. Intell. 31, 2179 (2009).

[22] N. Dalal, and B. Triggs, “Histograms of oriented gradients for human detection,” IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol.1, pp.886-893, 2005.

[23] K. Mikolajczyk, and C. Schmid, “A performance evaluation of local descriptors,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.27, no.10, pp. 1615-1630, Oct. 2005.

[24] S. Belongie, J. Malik, and J. Puzicha, “Shape matching and object recognition using shape contexts,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no.4, pp.509-522, April 2002.

[25] T. Ojala, M. Pietikäinen, and T. Mäenpää, “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, July 2002.

[26] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” International journal of computer vision, vol.60, no.2 pp. 91-110, 2004.

[27] H. Bay, T. Tuytelaars, and L. V. Gool, “Surf: Speeded up robust features,” Computer vision–ECCV2006, pp. 404-417, May 2006.

[28] P. Viola and M. J. Jones. Robust real-time face detection. International Journal of Computer Vision, 57(2), pp. 137–154, 2004.

[29] S. Zhang，C. Bauckhage，AB. Cremers, “Informed Haar-Like Features Improve Pedestrian Detection”, CVPR, 2014.

[30] X. Yuan，X. Shan，L. Su, “A Combined Pedestrian Detection Method Based on Haar-Like Features and HOG Features”, International Workshop on Intelligent Systems & Applications, 2011.

[31] Y. Wei，Q. Tian，T. Guo, “An Improved Pedestrian Detection Algorithm Integrating Haar-Like Features and HOG Descriptors”, Advances in Mechanical Engineering, 2013.

[32] Y. Li，W. Lu，S. Wang，X. Ding, “Pedestrian Detection Using Coarse-to-Fine Method with Haar-Like and Shapelet Features”, International Conference on Multimedia Technology, 2010.

[33] P. Viola and M. Jones, "Rapid object detection using aboosted cascade of simple features," IEEE Conference on Computer Vision and Pattern Recognition, pp. 511-518, vol.1, 2001.

[34] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, FPGA-based face detection system using Haar classifiers, In Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, pp.103-112, 2009.

[35] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.32, no.9, pp.1627-1645, Sep. 2010.

[36] C. H. Lampert, M. B. Blaschko, and T. Hofmann, “Beyond sliding windows: Object localization by efficient subwindow search,” IEEE Conference on Computer Vision and Pattern Recognition, pp.1-8, June 2008.

[37] B. Alexe, T. Deselaers, and V. Ferrari, “Measuring the objectness of image windows,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.34, no.11, pp.2189-2202, Nov. 2012.

[38] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, K. Doll, FPGA-Based Real-Time Pedestrian Detection on High-Resolution Images. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 629-635, 2013.

[39] K. Mizuno, K. Takagi, S. Izumi, H. Kawaguchi, 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. E96-C (4), pp. 433-443, 2013.

[40] Q. A. Zhu, M. C. Yeh, K. T. Cheng, and S. Avidan, Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2006, p. 1491.

[41] C.Wojek, G. Dorko, A. Schulz and B. Schiele, Joint Pattern Recognition Symp., 2008, p. 71.

[42] T. P. Cao and G. Deng, Proc. Int. Conf. Digital Image Computing: Techniques and Applications, 2008, p. 465.

[43] N. Dalal, Ph.D. thesis, Institut National Polytechnique de Grenoble (2006).

[44] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann and K. Doll, Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshops, 2013, p. 629.

[45] G. Overett, L. Petersson, N. Brewer, L. Andersson and N. Pettersson, A New Pedestrian Dataset for Supervised Learning, IEEE Intelligent Vehicles Symposium, 2008.

[46] P. Dollár, C. Wojek, B. Schiele and P. Perona, Pedestrian Detection: An Evaluation of the State of the Art, IEEE Trans. on Pattern Analysis and Machine Intelligence, 2012.

[47] P. Dollár, C. Wojek, B. Schiele and P. Perona, Pedestrian Detection: A Benchmark, IEEE Conference on Computer Vision and Pattern Recognition, 2009.

[48] M. Enzweiler and D. M. Gavrila, Monocular Pedestrian Detection: Survey and Experiments, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.31, no.12, pp.2179-2195, 2009.

[49] 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 (3), pp. 325-337, 2016.

[50] X. Zhang, F. An, L. Chen, et al., “Reconfigurable VLSI implementation for learning vector quantization with on-chip learning circuit,” Japanese Journal of Applied Physics, vol.55, no.4S, pp.04EF02:1-6, 2016.

[51] F. An, T. Akazawa, S. Yamasaki, et al., “VLSI realization of learning vector quantization with hardware/software co-design for different applications,” Japanese Journal of Applied Physics, vol.54, no.4S, pp. 04DE05:1-5, 2015.

[52] F. An, and H. J. Mattausch, “K-means clustering algorithm for multimedia applications with flexible HW/SW co-design,” Journal of Systems Architecture, vol.59, no.3, pp.155-164, March 2013.

[53] F. An, T. Koide, and H J. Mattausch, “A K-means-based multi-prototype high-speed learning system with FPGA-implemented coprocessor for 1-NN searching,” IEICE TRANSACTIONS on Information and Systems, vol.E95-D, no.9, pp.2327-2338, 2012.

CHAPTER 4: Reconfigurable On-chip Learning Coprocessors

Learning vector quantization (LVQ) neural networks have already been successfully applied for image compression and object recognition. In this study, firstly, I propose a dual-mode LVQ coprocessor featured dedicated learning circuits, enabling both on-chip learning and classification. The designed reconfigurable pipeline with parallel p-word input (R-PPPI) architecture was taped-out using 180 nm CMOS technology with parallel 8-word inputs and 102 K-bit on-chip memory. The prototype achieves low power consumption of 66.38 mW (at 75 MHz and 1.8 V) in an area of 7.89 mm^2. Secondly, I upgrade the 1st generation by a new modular and reconfigurable pipeline architecture (MRPA). The MRPA removes the dedicated learning circuits and expands the word-parallelism to 32 with 609 K-bit SRAM. The circuits consist of dynamically reconfigurable modules and realize a run-time and on-chip configuration for recognition and learning. In addition, the designed LVQ ASIC has high flexibility with respect to feature-vector dimensionality and reference-vector number, allowing the execution of many different machine-learning applications. Prototype fabrication in 65-nm CMOS technology achieves high-density efficiency and memory utilization efficiency with a core area of 2.14 mm^2, and average power consumption of 9.4 mW at 100 MHz and 0.8 V supply voltage. Compared with the embedded microprocessors, which rely on single-instruction-multiple-data (SIMD) processing, the developed prototype increases the performance of both recognition and learning operations. The MRPA prototype shows improvements by factors of approximately 40 and 101 on the well-established performance metrics million connections per second (MCPS) for recognition and million connection updates per second (MCUPS) for learning, respectively.

4.1 Overview for Leaning Vector Quantization (LVQ) Trainer and Classifier

Visual perception as one of the most advanced human capabilities is very difficult to achieve for artificial object recognition systems. Whereas, humans have the ability to detect and recognize thousands of objects in a scene with little or no conscious effort, despite changes in occlusions, illumination and the object's pose. Artificial neural networks (ANNs) are widely applied and very effective for pattern recognition [1, 2], function approximation [3], scientific classification [4, 5], control [6], and the analysis of time serial data [7]. Usually, ANNs have intrinsic units with massive vector-parallelism and a large number of interconnections among each other. Hardware ANNs based on conventional single instruction multiple data (SIMD-

mode) solutions, which help to achieve often necessary real-time response due to their parallel processing ability, have attracted increasing attention and have already been applied for color image compression [8], computation engines [9], robot locomotion control [10], multilayer perceptions [11], wind-speed sensor less control [12], olfactory systems [13], real-time object detection [14], and so on.

An neural network is a parallel and distributed network of simple nonlinear processing elements (PEs) or neurons interconnected in a layered arrangement [15]. Parallelism, modularity, and dynamic adaptation are three inherent characteristics of neural networks [16]. The parallelism of neural networks motivates much research, because the neural system has the potential to mitigate the computational limitations of serial SIMD architectures. However, most research relies on software which sequentially implements the neural networks. As a result, a software implementation is insufficient for many applications because of its weak performance. A hardware implementation can efficiently utilize the parallelism of neural networks, and therefore can outperform software implementations. Many hardware implementations of different neural networks have been presented previously [17]-[19].

Self-organizing-map (SOM) neural network models, which were introduced by Willshaw et al [20]. and Kohonen [21], have been used in a wide variety of fields such as unsupervised learning tasks [22], data exploration [23], and water resource exploration [24]. As an unsupervised vector quantization method, the self-organizing map (SOM) is closely related to LVQ. LVQ was introduced by Kohonen [25] as a family of intuitive, universal and efficient multiclass classification algorithms. There have been many applications of LVQ, such as in handwriting recognition [26], odor recognition [27], medical biology [28], economical optimization [29], and alertness detection [30].

The learning process of LVQ is intuitively clear and classification decisions are based on the nearest neighbor search (NNS) among the reference vectors, also called neurons as well. In general, learning in the LVQ algorithm is realized by modifying the reference-vector values according to a distance function and the input-vector matching results, thus representing a process of approximating the theoretical Bayes decision borders. The winner-reference vector, which is most similar to the input vector, is adjusted towards the input vector, if their classes are the same. Otherwise, the winner-reference vector is moved away from the incorrectly classified input vector. At the beginning of the learning process, reference vectors at some initial positions are randomly selected. Then, the input vectors for the learning process are sequentially processed and the values of reference vectors are continuously updated to increase the LVQ accuracy.

In the literature, SOM and LVQ were implemented off-line in software on computer systems or embedded processors. The first reported implementation of SOM is a software implementation on a processor [31]. In [32], an LVQ implementation with 605 weight vectors takes about 0.56 seconds for recognition. Such software solutions are flexible, easy to implement, and are often designed in advance of a hardware implementation to help promptly make rational design choices during the exploration phase. The performance of single-core microprocessors has been improved by the multi-core parallelism in multiprocessor system-on-chip (MPSoC) [33]. In addition, researchers have investigated the application of graphics processing units (GPUs) for accelerating the training by exploiting the parallel and high-precision computing capability of GPUs. In [34] a heterogeneous computing model for LVQ is presented. The applied method in [34] requires memory transfers between the central processing unit (CPU) and the GPU's global memory, because the weight vectors and input vectors are stored separately in the GPU and the CPU. The authors of [34] have executed the CPU implementation on a Xeon X3440 sever with 2.53 GHz clock frequency, while the GPU implementation was executed on GTX 680 system. A performance of 54731.3 MCUPS was achieved at 2.53 GHz for 3755 160-dimensional weight-vectors. However, the processor-based architectures are still suffering from a large degree of sequential processing and from high power consumption, when compared to application-specific solutions. For real-time applications, these software-based approaches can therefore not deliver sufficient performance for online learning due to the high cost of the computational-requirements. Consequently, a number of methods and techniques were proposed to implement LVQ or SOM in hardware [34]-[37].

4.2 Previous Work on LVQ Coprocessors

Most hardware research on LVQ concentrated on making optimal use of the parallelism by increasing the possible number of PEs or neurons in limited hardware resources, but did not explore the modularity and dynamic adaptation of neural networks. Considering that LVQ algorithms are "multiplication-rich," and that the hardware cost of a multiplier is very high, a static-configuration hardware often leads to resource shortage and waste. More hardware than available may be required, whereas hardware realizing the same functions such as multiplication during different processing phases cannot be re-exploited.

In this regard, Field Programmable Gate Array (FPGA) implementations of LVQ have attracted a lot of attention [38]-[43]. However, a FPGA represents a fine-grained reconfigurable architecture, which often has low efficiency. Owing to its poor routability, the routing area

overhead of an FPGA is normally quite large. Extensive usage of fine-grained reconfigurable logic to perform calculation-rich applications demonstrates a favorable tradeoff between flexibility and performance, as shown in Fig. 4-1 [44]. In general, the more flexible a machine is, the simpler the programming is, but the lower the performance becomes. Special-purpose hardware with coarse-grained reconfiguration can offer a flexibility closer to instruction-set architectures and at the same time achieve high performance near to that of fully customized hardware. In addition, a coarse-grained reconfigurable ASIC can achieve high area efficiency while maintaining low placement and routing complexity. Considering the decomposition method of a complicated function into sub-functions, there is no necessity to activate all the sub-functions at the same time. Through rapid reconfiguration, a rather small piece of hardware can thus realize multiple functions, required at different processing stages.

A pipeline architecture can accelerate and simplify the reconfiguration process, because the implementation is piece-wise, which can massively reduce reconfiguration time [45]. In general, increasing the amount of hardware pieces can proportionally improve the pipeline's performance. A schematic diagram of the flexibility-performance comparison between the traditional approaches and our research, which exploits coarse-grained reconfiguration, is given in Fig. 4-1.

The SIMD-based solutions with their massive parallelism have attracted much attention to implementing LVQ and SOM. The well-established performance metrics MCPS (Million Connections per Second) and MCUPS (Million Connection Updates per Second) are separately used to evaluate recognition and learning modes of the designed chip.

In [46], a vision chip was fabricated in 180-nm CMOS technology, where a SIMD processor could be reconfigured as a 16×16 SOM neural network, which consumed 40.8% total area (33.6

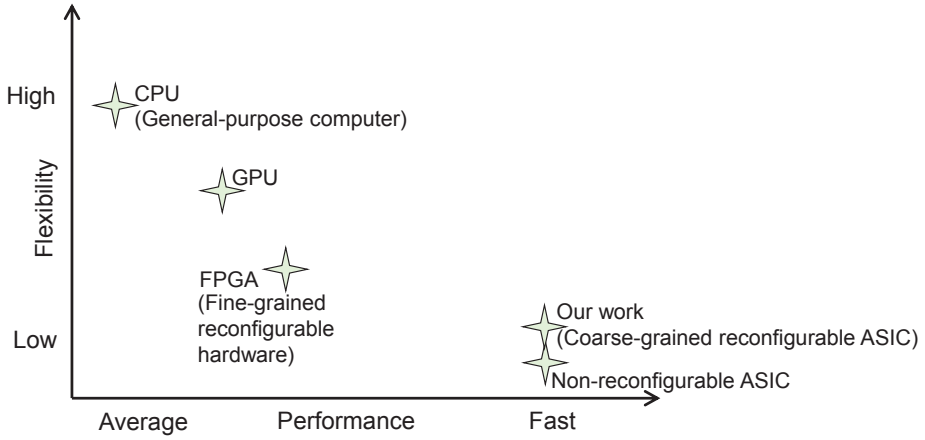


Fig. 4-1. Schematic of the flexibility and performance target of the reported coarse-grained reconfigurable and pipelined ASIC architecture.

mm^2). The SOM neural network was trained online through the LVQ method. The estimated power dissipation was about 257 mW because the power dissipation is proportional to the relatively large chip area. Recognition performances of 186 MCPS and 258 MCPS for 16-dimensional and 32-dimensional vectors, respectively, were achieved at 50 MHz.

In [48], the authors implemented LVQ on FPGA through the selection of the best option among a number of architectures produced by FPGA software design tools. Dedicated learning and recognition circuits are needed. The vector dimensionality is fixed to 23 with 16-bit precision. The XC3S1400AN-based prototype achieved 23.95 MCPS when it was working at 50 MHz with 350 weight-vectors.

An algorithm architecture adequacy methodology for LVQ implementation was proposed in [49]. A performance of 11.29 MCUPS and 136.95 MCPS was achieved on a Xilinx XC4VLX100 FPGA working at 50 MHz with 12 21- dimensional weight-vectors and 14-bit precision.

In [50], a sequential/parallel architecture for LVQ was presented. The vectors were sent in series to the neuro-processors which were operated in parallel. The attained performance on a Xilinx XCV1000 FPGA were 1115.84 MCPUS and 1543.83 MCPS at 100 MHz while processing 49 23-dimensional weight-vectors with 8-bit precision.

In [51], the authors adopted on-line serial arithmetic operators for LVQ implementation. The best performance with a Xilinx XCV1000E FPGA was 625 MCPS at 25 MHz while processing 25 23-dimensional weight-vectors at 8-bit precision.

An accuracy extension of the algorithm architecture adequacy methodology for LVQ was implemented in [52]. The learning performance on a Xilinx XC4LX25 FPGA reached 6.25 MCUPS at 25 MHz while processing 25 23-dimensional weight-vectors at 8-bit precision.

The authors in [53] implemented LVQ on an Altera ACEX1k100 device with 32 neurons and 16-bit precision. The learning was done off-chip and Manhattan distance was used. The recognition performance reached 6.9 MCPS at 25 MHz while processing 64-dimensional vectors.

Instead of the massive parallelism, we propose a modular and reconfigurable pipeline architecture for accelerating the LVQ algorithm. This proposed architecture can achieve good performance in both recognition and learning, high integration density and memory-utilization efficiency. Further, the reported work adopts the Euclidean distance metric which often provides higher accuracy in practical applications than the Manhattan distance used in [41], [43] and [46].

4.3 LVQ Algorithms

LVQ is a supervised-learning neural network and is popular for nearest-neuron-based recognition, especially multiclass recognition [21]. The LVQ neural network consists of three layers, which are input layer, hidden layer, and output layer. An *M*-dimensional vector, called a weight vector (*w*), is assigned to every neuron in the hidden layer. A winner-takes-all (WTA) mechanism determines the winner vector (*ws*), which is the weight vector having the minimal distance to the input vector. The interconnections between input layer and hidden layer are dynamically adaptable to realize learning and recognition. The input layer is problem-dependent, so that *M* varies for different applications. For example, if we use Histogram of Oriented Gradients (HOG) features as input vectors, *M* is 3780. The hidden layer calculates either the distances between the input vector and weight vectors or the adapted *ws*.

Corresponding to the learning and recognition of a LVQ neural network, the LVQ algorithms have two operations, learning and recognition. The learning operation includes search and adaptation of the *ws*, which is determined by a distance metric, e.g. the Euclidean distance (*ED*). After learning, the weight vectors remain unchanged for recognition. The notations that are used in the remaining paper are listed in Table 4-I for convenience.

Suppose that *x(t)* and *ws(t)*, which are *M*-dimensional vectors, respectively, represent the input and winner vector in the discrete-time domain. Correspondingly, *vx* and *vs* are the class labels of *x(t)* and *ws(t)*. Furthermore, *a* represents the learning rate. In the learning mode, *ws(t)* is adapted to better comply with *x(t)* according to Step 3 of the learning process listed below.

Step 1: Randomly initialize the weight vectors to *v* classes and set the learning rate *a*.

Step 2: For one labeled input-vector *x(t)*, search its *ws* by nearest-neighbor-search (NNS).

Step 3: Adaption of *ws* based on the label-comparison result. If *x(t)* and *ws(t)* belong to same class, i.e., *vx* is equal to *vs*, *ws(t)* is moved closer to *x(t)* and the new value of the winner vector (*ws(t+1)*) becomes:

$$ws(t+1)= ws(t)+a[x(t)- ws(t)] \tag{4-1}$$

Otherwise, the new value of the winner vector becomes:

$$ws(t+1)= ws(t)-a[x(t)- ws(t)] \tag{4-2}$$

Step 4: Repeat Steps 2 and 3 until reaching either a threshold or other termination conditions.

As shown in [26], the mean-square error is defined as:

$$E = \int ||x - ws||^2 p(x) dVx \tag{4-3}$$

Where dVx is a volume element in the x space and $p(x)$ is the probability density function, which defines the statistical frequency for occurrence of the samples $x(t)$ in eq. (4-1). The

Table 4-I. Definition of Notations

Symbol	Definition and comments
x	Input vector
J	The number of input vectors
w	Weight vector of each neuron
ws	Winner vector
ws(t)	Old winner vector
ws(t+1)	Adapted winner vector
v	The number of classes
vx	Class label of input vector (in learning mode)
vs	Class label of winner vector
a	Learning rate
M	Vector dimensionality
NN	The number of neurons (weight vectors)
N	Word-parallelism/the number of parameterizable-storage modules/the number of weight modules /the number of elementary-adder-modules, *N* is 32 for the prototype
P	The number of partial vectors/ local pipeline-stages of summation module during the adaption phase because of the partial storage, $P=\lceil M/N \rceil$ is the smallest integer not less than *M/N*. $P \geq 1$
G	Physical pipeline-stages without partial storage, $G=log_2N+6$
L	Local pipeline-stages of summation module during the nearest-neighbor-search phase because of the partial storage, $L=log_2N+P$
c	Stage of a static pipeline
f	Stage of a reconfigurable pipeline which realizes equivalent functionalities as in the *c*-stage static pipeline
F	Working frequency
O	Computational complexity

weight vectors w are optimally placed when E is minimized. In particular, the minus sign in eq. (4-2) defines corrections corresponding to the subtraction of a fraction $p(x)$ of the neighboring (overlapping) class from the class to which ws belongs. As a result, the difference of $p(x)$ for the neighboring classes falls to zero at the class borders. This means the LVQ algorithm tends to pull the weight vectors away from the class borders [49]. In the recognition mode, the *x(t)* is unlabeled and is assigned to the same label as its *ws(t)*.

4.4 Hardware Architecture

4.4.1 Modular and Reconfigurable Pipeline Architecture (MRPA)

Traditionally, the massively parallel approaches, e.g. SIMD, comply with the definition of neurons in neural networks. Hence, each neuron corresponds to a PE, which contains a weight-memory block and ALUs to store and process one weight-vector. The maximum dimensionality of the weight vectors is limited by the size of individual weight-memory block. In our work, we segregate the weight-memory blocks from the PEs as a shared memory-pool. All the weight vectors share the same ALUs rather than having individual ALUs.

Overall, the proposed MRPA with N word-parallelism consists of one control unit (CU) and four specific function modules (SFMs), which are the parameterizable-storage modules (PSMs), weight modules (WMs), summation module (SM), and comparison module (CM), as shown in Fig. 4-2. Rather than mapping each neuron to a dedicated PE, the MRPA shares its specific function modules with all neurons. Further, the MRPA employs a concept of partial processing and divides an M-dimensional vector into P N-dimensional components ($P=\lceil M/N \rceil$ is the smallest integer not less than M/N). The specific function modules have pipeline registers, which synchronously latch the data with the same rising clock edges, so that each computed value can be latched in its following register. That means, the MRPA is a parallel-pipeline cascaded system, comprising of N parallel input ports and G physical-stage pipelines ($G=log_2N+6$). Moreover, weight modules, summation module and comparison module have dynamical reconfigurability and realize multiple functions in different phases. The architectural structure, consisting of parameterizable-storage modules, weight modules and summation module, not only calculates the squared Euclidean distance (ED^2) between input and weight vectors but also adapts the winner vectors. The MRPA performs partial configuration at run-time during the adaption mode. Firstly, the MRPA is configured to search for the weight-vector with the minimal ED^2 distance to the input sample. Then, weight modules, summation module

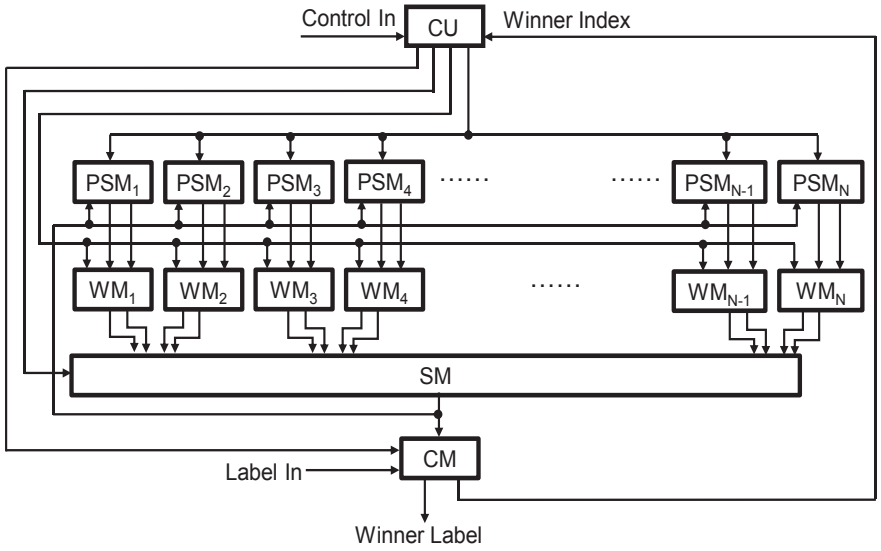


Fig. 4-3. Modular architecture for LVQ with N word-parallelism.

and comparison module are dynamically configured [50] to update the best-matching weight vector without changing or stopping the other parts. The dynamic reconfiguration ensures that the MRPA can temporally partition the algorithms and time multiplex the logic to meet the hardware resource constraints. The MRPA reconfigures the logic at the run time, i.e., when parts of the logic (weight modules, summation module and comparison module) are replaced, while other active circuits (parameterizable-storage modules) operate uninterrupted. Furthermore, the MRPA offers easy scalability, and can effectively adjust the parallelism and communication infrastructure.

More specifically, the control unit provides a communication infrastructure between the specific function modules and includes a register array which stores external control signals, local control signals, and local feedback signals, as depicted in Fig. 4-3. During run-time, the signals "Mode" and "Comparison Result" configure the data path of weight modules, summation module and comparison module. The parameterizable-storage modules employ a partial-storage concept to store vectors [51] and assign P addresses of N memory blocks to each M-dimensional vector (see Fig. 4-4). Each weight module realizes the multiplication function to calculate the 1-dimensional squared Euclidean distance $[x(t)_i\text{-}w(t)_i]^2$ or the 1-dimensional correction value ($\pm\alpha\ [x(t)_i\text{-}ws(t)_i]$) ($i \in [1, N]$). The summation module accumulates the partial squared Euclidean distance, or corrects the partial winner-vector. The comparison module searches the winner vector through comparing the squared Euclidean distances and determines the sign of the learning rate α or labels the input vector.

The learning mode of LVQ has two phases, namely nearest-neighbor-search phase and

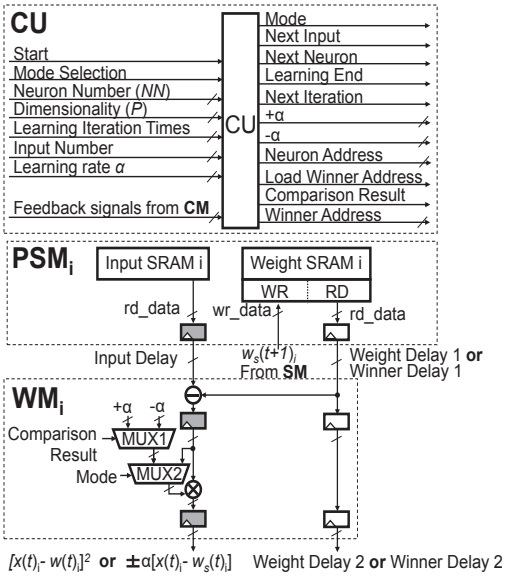


Fig. 4-5. Details of the control unit (CU), the parameterizable storage module (PSM) and the weight module (WM).

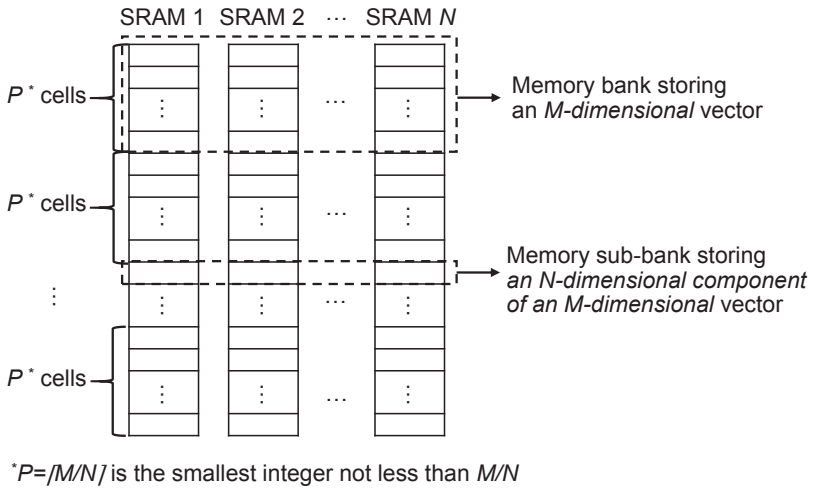


Fig. 4-7 The partial storing concept applied to vector storage. An M-dimensional vector occupies P cells of the N SRAMs.

adaption (updating) phase, while the recognition mode only needs the nearest-neighbor-search phase. During the nearest-neighbor-search phase, the MRPA searches the winner vector through the squared Euclidean distance comparison between the input vector and the weight vectors. First, the weight modules calculate the squared differences of partial input and weight-vectors read from the parameterizable-storage modules. Then the summation module accumulates the partial squared Euclidean distance results from the weight modules and will not transmit the accumulation result to the comparison module until the summation module completes the processing of all the *P* components of one weight-vector. The comparison module compares the currently transferred squared Euclidean distance with the local minimum squared Euclidean distance.

When searching for the whole set of weight-vectors completes, the comparison module selects the sign of α while in the learning mode or labels the input vector while in the recognition mode. During the adaption phase, the MRPA solely updates the winner vector and keeps the other weight-vectors invariant. The winner vector is located at the "Winner Address," which is transferred from the comparison module to the control unit as shown in Fig. 4-3. The adaption scheme follows eq. (4-1) when input and winner vector belong to the same class, or eq. (4-2) when the input and winner vector have different labels. The sign of α reflects the adaptive direction and relies on the "Comparison Result" signal in the comparison module. The weight modules compute the correction value, which is the multiplication between α and the difference of the input vector and winner vector ($\pm\alpha[x(t)-ws(t)]$). The summation module adds the correction value to the winner vector and transmits the adapted winner to the parameterizable-storage modules, where the old winner *ws(t)* is overwritten with the adapted winner-vector

ws(t+1). *N* parallel weight SRAMs in the parameterizable-storage modules concurrently write the *N* components of the adapted winner vector at the "Winner Address".

Critical MRPA operations can be summarized as follows:

1. Initialize the parameters of the control register array, e.g. for mode configuration, dimensionality, and number of weight vectors.
2. Initialize the weight vectors with prepared data.
3. For the learning mode, the control unit configures to the nearest-neighbor-search phase at first, and then switches to the adaption phase. The recognition mode only undergoes the nearest-neighbor-search phase.

4.4.1.1 Control Unit (CU), Parameterizable Storage Module (PSM) and Weight Module (WM)

The control unit (see Fig. 4-3) is composed of a finite state machine, a control register array, and several counters. The control register array stores the predefined variables, local control signals, and local feedback signals. The predefined variables include the number of weight vectors, vector dimensionality, learning rate α, *P*, learning iteration times, and external control signals. The counters decode the external variables to dynamically changeable local control signals. For example, the signals "Next Neuron" and "Next Input" depend on the number of weight vectors. E.g., when the vector-component counter meets the predefined *P*, the "Next Neuron" signal indicates that the squared Euclidean distance calculation between the input and next weight-vector will start. When the neuron-number counter reaches the predefined number of weight vectors, the "Next Input" signal asserts to request the next input-vector. The counters contribute to the easy parameterization of MRPA regarding the number of weight vectors, vector dimensionality, and learning iterations. The flexible setting of learning rate α and learning iterations further contributes to the flexibility of adaption strategies. The external control signal "Mode" selects the appropriate mode of operation: learning or recognition. The "Comparison Result" signal is a local feedback signal from the comparison module which indicates the label-comparison result between the input and its winner-vector.

A parameterizable-storage module (see Fig. 4-3) consists of a single-port SRAM for input vectors, a dual-port SRAM for weight vectors, and two pipeline registers. The *N*-parallel parameterizable-storage modules split an *M*-dimensional vector into *P* *N*-dimensional components (see Fig. 4-4). Therefore, an *M*-dimensional vector occupies *P* addresses of *N* SRAM blocks. "RD" and "WR" in Fig. 4-3 represent data-read port and data-write port of the dual-port SRAM, respectively. During both the nearest-neighbor-search phase and adaption

phase, the weight vectors are read out from the "RD" port. The "WR" ports are only activated during the adaption phase when writing the adapted winner-vector back to the location of old winner. All the weight vectors share a memory pool consisting of the N dual-port SRAMs instead of having individually fixed memory space. The sharing scheme satisfies the unfixed space requirements to parameterize both vector dimensionality, and number of weight vectors. Each weight module attaches to one parameterizable-storage module and comprises a subtractor, two multiplexers, a multiplier, and four pipeline-registers, as shown in Fig. 4-3. The multiplexer MUX2 decides the function of the weight module, which is a 1-dimensional squared Euclidean distance calculation $[x(t)_i\text{-}w(t)_i]^2$ or a 1-dimensional correction value computation ($\pm\alpha[x(t)_i\text{-}ws(t)_i]$). The N weight modules deliver their outputs to the summation module (see Fig. 4-5).

4.4.1.2 Summation Module (SM) and Comparison Module (CM)

The summation module contains N elementary-adder-modules (EAMs) and a multiplexer (MUX3), and summarizes the operands from the weight modules, as depicted in Fig. 4-5. An elementary-adder-module contains two identical multiplexers (MUX4), one adder, and one register. The multiplexers MUX4s reconfigure the summation module to realize multiple functions, which are calculation and accumulation of the partial squared Euclidean distance, or a correction of the partial winner-vector. The MUX4 pair in each elementary-adder-module passes the corresponding two operands to the adder during different phases.

During the nearest-neighbor-search phase, the summation module constructs the elementary-adder-modules as a complete binary tree with the same length of each signal path (see Fig. 4-

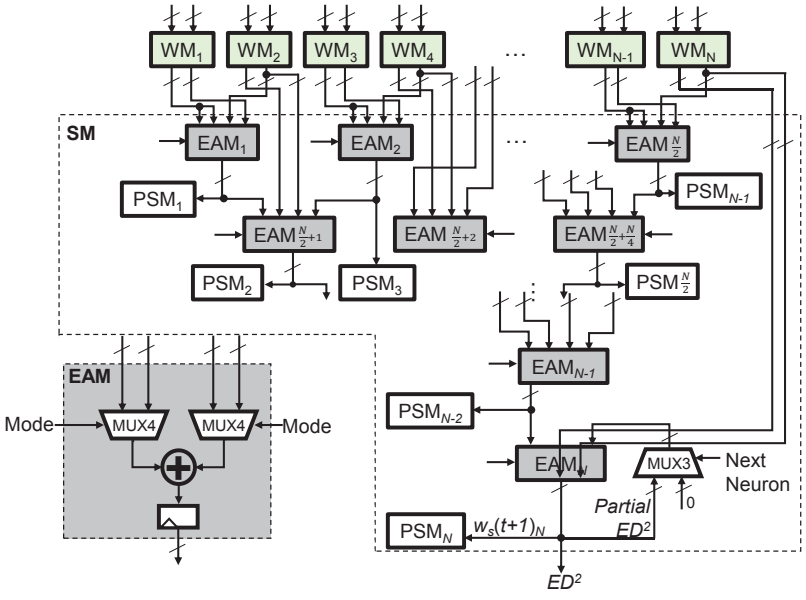


Fig. 4-9. Detailed construction of the summation module (SM) and the elementary adder module (EAM). The SM includes N EAMs.

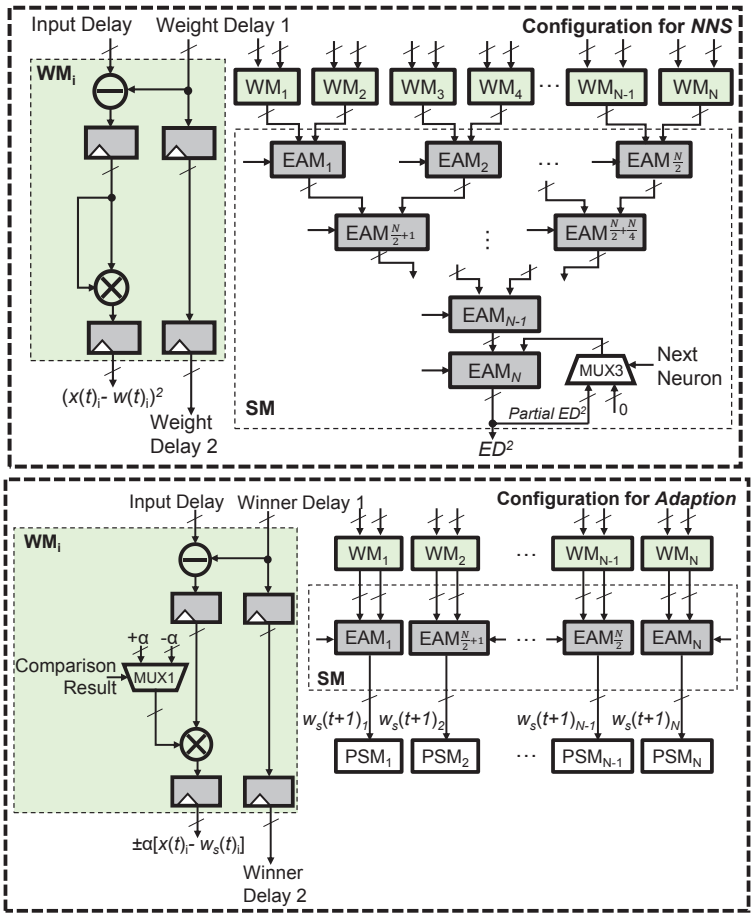


Fig. 4-11. The topological structure of weight module (WM) and summation module (SM) with implemented dynamic reconfiguration capability for the phases of nearest neighbor search (NNS) and winner-vector adaption.

6), instead of as *N*-parallel adders with high latency and large Si-area. Owing to the implementation of the complete binary tree and the partial processing, the required number of elementary-adder-modules (NUM_{EAM}) solely depends on the word-parallelism of MRPA. Eq. (4-3) is the formula to calculate the NUM_{EAM} for the *N*-parallelism MRPA. NUM_{EAM} is the sum of a geometric progression plus one. The geometric progression results from the complete binary tree architecture and the "one" follows the partial processing. The log_2N refers to the number of levels in the complete binary tree. The last elementary-adder-module (EAM_N) accumulates the *P* partial-squared Euclidean distances. The MUX3 only works during the nearest-neighbor-search phase and behaves like a switch, whose turn-on signal is "Next Neuron". When the EAM_N has accumulated all the *P* partial- squared Euclidean distances between one pair of input and weight vector, the MUX3-switch turns on. Then the summation module outputs the completed squared Euclidean distance to the comparison module. Simultaneously, the summation module initializes the partial- squared Euclidean distance as "0".

$$NUM_{EAM} = \sum_{k=1}^{\log_2 N} \frac{N}{2} \cdot \left(\frac{1}{2}\right)^{k-1} + 1 = N \quad (4\text{-}4)$$

During the adaption phase, the summation module reconfigures the elementary-adder-modules as a distributed interconnect-structure (see Fig. 4-6). Each elementary-adder-module attaches to one weight module and one parameterizable-storage module without redundancy or deficiency of modules.

Owing to the two different configurations, the summation module is implemented with different local pipeline-stages, each of which takes one clock cycle. During both the nearest-neighbor-search and adaption phases, the summation module reuses the physical-stages *P* times owing to the partial processing. For example, during the nearest-neighbor-search phase, the summation module takes *L* (*L* =log_2N+ *P*) clock cycles and implements the *L* local-stages with log_2N+1 physical-stages. During the adaption phase, the summation module uses *P* clock cycles and realizes the *P* local-stages using one physical-stage.

The comparison module, shown in more detail in Fig. 4-7, includes two logic comparators, five registers, and one AND gate. The comparison module operates during the nearest-neighbor-search phase and accomplishes different tasks for learning and recognition modes. In the learning mode, the comparison module searches and outputs the winner index to the control unit for the following winner-adaption. The winner index includes "Comparison Result" and "Winner Address". As for the recognition mode, the comparison module outputs the "Winner Label" as the result. During both modes, the squared-Euclidean-distance comparator determines the winner by comparing the newly calculated squared Euclidean distance with the local minimal squared Euclidean distance. The label comparator that only works during the learning mode compares the label of the input vector with its winner vector and outputs the "Comparison Result" signal to the control unit. When the winner label is equal to the input label, the weight modules select +α; otherwise, the weight modules select -α.

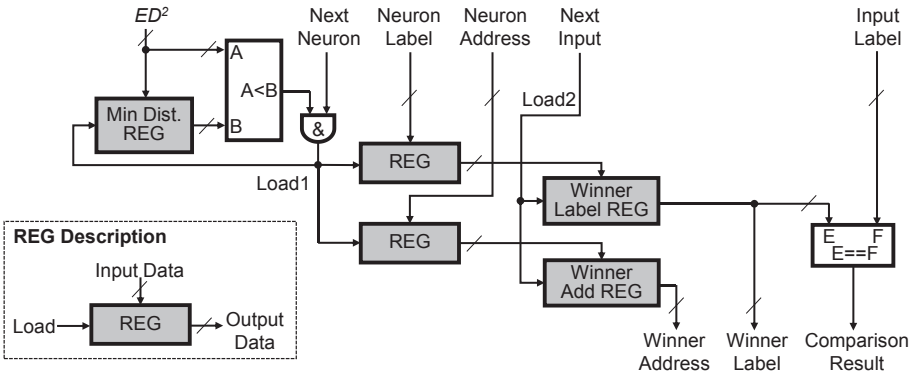


Fig. 4-13. Schematic of the comparison module (CM) to find the winner vector. The registers have a load enable signal.

4.4.1.3 Pipeline Reconfiguration, Modularity and Parameterization

The MRPA embodies the pipeline reconfiguration and modularity using inter-stage parallelism and intra-stage parallelism.

The inter-stage parallelism is enhanced by partitioning a complex function into six independent sub-functions which are executed in a serial and synchronized fashion. We mapped the six sub-functions to the four specific function modules (see Fig. 4-10) rather than applying a specific hardware-unit for each sub-function. Two of the specific function modules (weight modules and summation module) attain functional reconfiguration [52] and realize multiple sub-functions (see Fig. 4-6). The time-multiplexed units (MUX2 and MUX4 in Fig. 4-3 and Fig. 4-5) configure the data paths of weight modules and summation module for nearest-neighbor-search and adaption. In addition, the MRPA achieves run-time reconfiguration, which ensures the multiple sub-functions realization on the same hardware in the discrete time-domain. Moreover, the reconfiguration process is completely on-chip by applying a time-multiplexing concept. The communication cost including the effort for transmitting initialization data, training data, and input data to the external memory from the host computer is drastically reduced. As a result, the MRPA accelerates the operation by eliminating the extra reconfiguration time and avoiding the reloading of data to the pipeline. For example, in the learning mode, the MRPA implements nearest-neighbor-search in the beginning, and then automatically switches to the winner adaption. Moreover, except for a few additional multiplexers, no particular vector-registers or other additional-operators are required for the reconfiguration.

The intra-stage parallelism is realized by dividing the N-dimensional operation into N 1-dimensional sub-operations that can be performed independently in parallel. The 1-dimensional vector processing matches with individual parameterizable-storage module, weight module, and elementary-adder-module. The modularity of 1-dimensional vector-processing enables the easy scalability of intra-stage parallelism in both upward- and downward-compatible fashions for future soft-IP designs.

Another important advantage of the MRPA is that it supports architecture parameterization which enhances its application areas without any changes in the ASIC hardware. The number of weight vectors, vector dimensionality, learning-iteration times, and α are all adjustable parameters.

4.4.2 Dedicated On-Chip Learning Circuits for Reconfigurable Pipeline with Parallel P-word Input Architecture (R-PPPI)

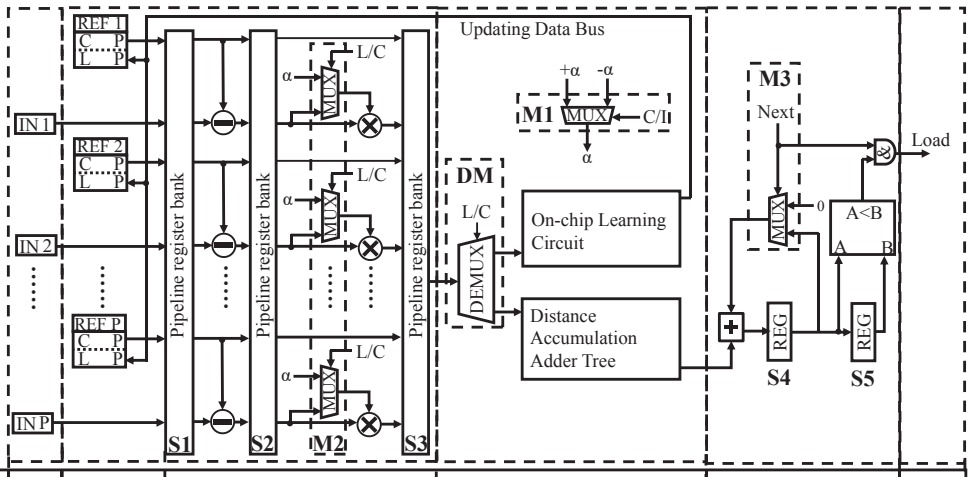
The section (1) explains the optimized architecture (MRPA) for LVQ. In this section, the fundament difference of the 1st and optimized editions is demonstrated, which is dedicated on-chip learning circuits. I eliminated the dedicated on-chip learning circuits in MRPA through modular reconfiguration, which saves both area and power.

The dual-mode system implemented by the R-PPPI architecture can switches between on-chip learning and classification mode of the LVQ. In the architecture for the p-parallel module of the R-PPPI shown in Fig. 4-8, the data path for the two modes is configured according to the signal “L/C”.

As described above, the on-chip learning procedure is realized by the R-PPPI architecture which has dual-mode capability configurable by the signal “L/C”. The learning time in clock cycles can be defined as in (4.5), where R is the reference number, ⌈d/p⌉ is the partial storage parameter and d is the vector dimensionality. The parallelism p of R-PPPI is a power of 2, namely 2y. The number “3” in eq. (4-5) represents the pipeline delays of registers S1, S2 and S3. The register S1 separates the memory blocks of the input layer from the subtractors. S2 is located between the subtractors and the multipliers, and S3 is between the multipliers and the adders. The parameter PD is the pipeline depth defined in (4.6). In particular, the first "2" is the pipeline delay of S1 and S2, while the pipeline delays due to S4 and S5 are reflected in the second “2” of (4.6).

$$Tlearning=(R+1)\times\lceil d/p\rceil+PD+3 \quad (4\text{-}5)$$

$$PD=2+y+2 \quad (4\text{-}6)$$



Part / Function	Input Layer	Competition Layer: Weight Unit	Competition Layer: Summation Unit	Winner-takes-all Part	Output Layer
Learning	Neuro Storage	$[x(t)- w_s(t)]a$	$w_s(t+1)= w_s(t)+a[x(t)- w_s(t)]$	Non	Non
Classification	Neuro Storage	$[x- w_i]^2$	$D_E^2=\sum_{i=1}^{n}(x_i-w_i)^2$	Distance Comparison	Label

Fig. 4-15. R-PPPI architecture for a memory-based LVQ neural network. The same hardware parts are configured to have different functionality in different operating modes of learning and recognition.

In the case of the fabricated test chip which has 8 parallel inputs, an on-chip learning step with one input vector needs (R+1)×⌈d/8⌉+10 clock cycles. Indeed, the learning efficiency has been improved to a much higher factor than for the conventional solutions even though the reference number R and the vector dimensionality d still have some limited effects. The comparison of the learning efficiency to the general purpose processor (Intel® Core™ i7) and the SoC solution [28] for pedestrian detection with 3780-dimensional HOG feature is illustrated in Fig. 4-9, where 2416 positive samples and 12180 negative samples in INRIA dataset [30] are used to train the LVQ references. The learning time with different of reference-vector numbers and the speedup factor to the software implementation demonstrate the very high learning efficiency that make online machine learning possible. Through applying a larger capacity memory, the designed LVQ ASIC can be extended to deal with much larger dimensional vectors and larger reference-vector numbers.

As shown in Fig. 4-9, the hardware implementation remarkably outperforms the software implementation on a PC with an advanced 3.40GHz Intel® Core™ i7-4770 CPU and 8 GB of RAM memory as well as the SoC solution [28] with a low power RISC CPU. In addition, the larger the number of reference vectors is, the larger speedup factor becomes. When the reference-vector number reaches 1000, the speedup factor is nearly 200 times. For LVQ algorithms, the accuracy increases with larger numbers of reference vectors. Apart from the much faster learning speed than in the software implementation, this work with much lower

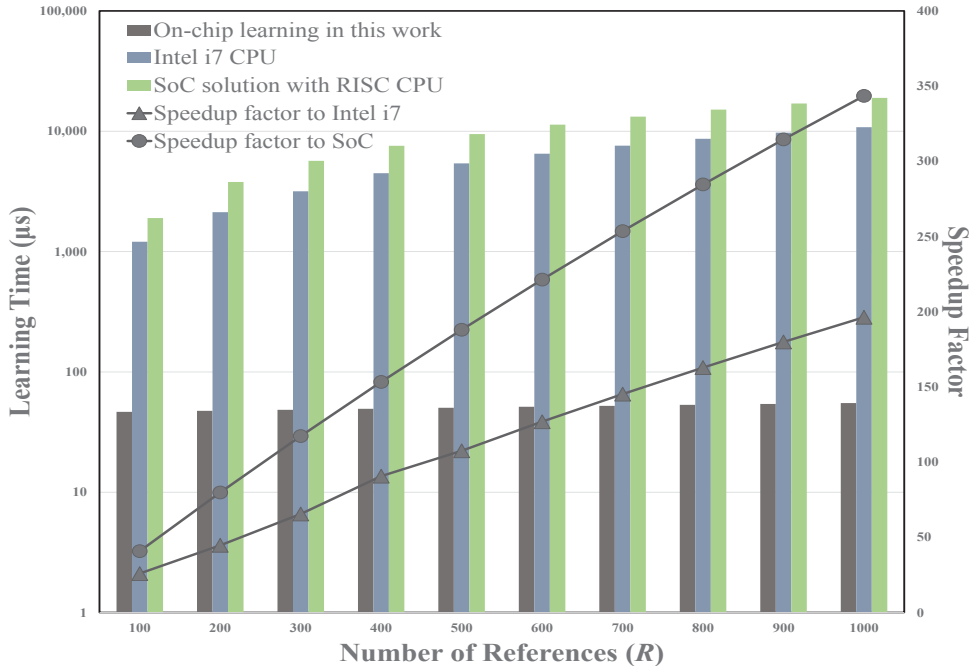


Fig. 4-17. Speedup factor in comparison to a software implementation using a 3.40GHz Intel® Core™ i7-4770 CPU, and a SoC solution[28] with a low power RISC CPU.

power dissipation also has very high energy efficiency. Although this work has somewhat lower flexibility than the general purpose CPU, the proved extendibility in vector dimensionality and reference-vector number allows to handle most of the real-world applications.

4.5 Implementation and Results

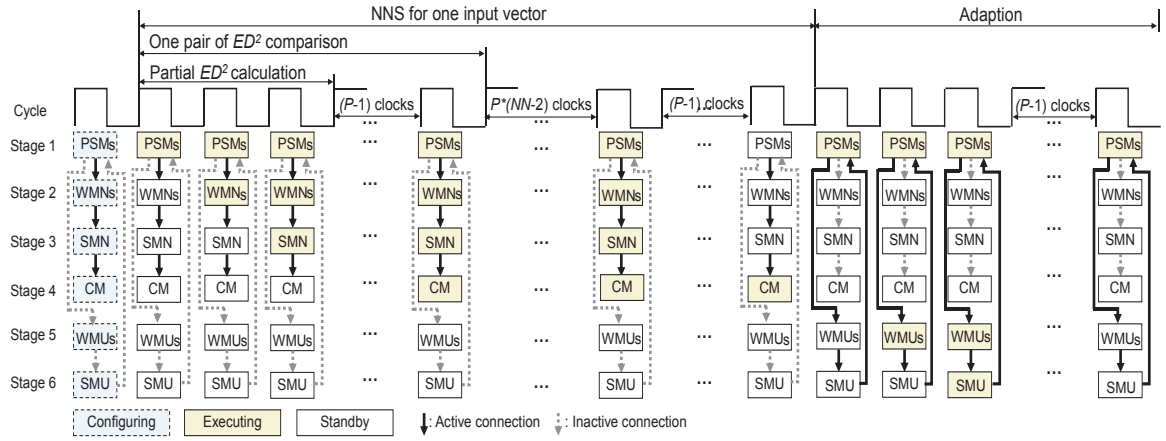
4.5.1 Performance Analysis

4.5.1.1 Density Efficiency

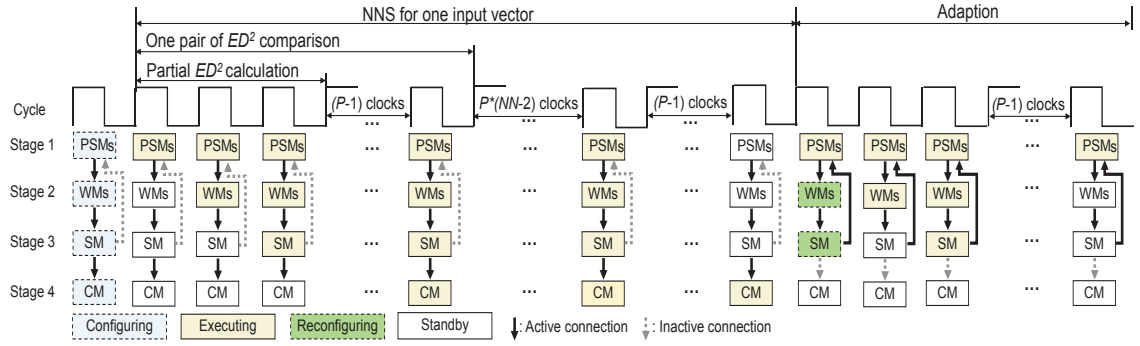
Density efficiency refers to the number of functions per unit area. The MRPA reconstructs the modules to extend its functions without extra hardware consumptions. The ALUs, such as multipliers, adders, and subtractors, which occupy large Si-area in hardware implementation, serve multiple functions. First, learning and recognition modes share the same hardware. Second, the MRPA implements the sequential back-propagation learning steps of the LVQ algorithm by dynamically reutilizing the ALUs to realize the sequence of nearest-neighbor-search and adaption phases in each learning step [53].

To ensure dynamic reconfiguration, the MRPA breaks a sequential processing-cycle into a series of stages which achieve inter-stage parallelism. All the stages are independent and executed in parallel. As shown in Fig. 4-10, the MRPA implements a six-stage static pipeline on a four-stage reconfigurable pipeline. All the specific function modules in Fig. 4-10(a) are static and have a dedicated sub-function whereas the weight modules and summation module in Fig. 4-10(b) are reconfigurable. Inside either the six-stage or the four-stage pipeline, there are local-stages to support the sub-functions of specific function modules. Each local-stage takes one clock cycle. For example, stage 3 of both static and reconfigurable pipelines in Fig. 4-10 has L ($L = log_2 N + P$) local-stages. Moreover, stages 2 and 4 of both static and reconfigurable pipelines, and the additional stages 5 and 6 of static pipelines each have two local-stages. To explain the reconfigurable pipeline more simply, we assume that every specific function module lasts one stage and ignore the clock-cycle differences within the specific function modules in Fig. 4-10.

Without pipeline reconfiguration, two additional specific function modules are necessary for the adaption phase, which are the weight-modules for nearest-neighbor-search (WMNs) to calculate $\pm\alpha[x(t)-ws(t)]$ and the summation-module for updating (SMU) to compute $ws(t+1)$ (see Fig. 4-10(a)). The weight modules in Fig. 4-10(b) can accomplish the sub-functions of weight-modules for nearest-neighbor-search (WMNs) and WMUs in Fig. 4-10(a). In a similar way, the summation module in Fig. 4-10(b) corresponds to the summation-module for nearest-



(a) Pipeline stages without reconfiguration



(a) Pipeline stages with reconfiguration

Fig. 4-19. Implementation of a 6-stage static pipeline on a 4-stage reconfigurable pipeline: (a) 6-stage pipeline without reconfiguration and (b) 4-stage pipeline with reconfiguration. To explain the reconfigurable pipeline more simply, we assume that every specific function module lasts one stage and ignore the clock-cycle differences within the specific function modules.

neighbor-search (SMN) and SMU in Fig. 4-10(a). The additional WMNs and SMU demand extra area compared with the MRPA. The pipeline reconfiguration in the MRPA utilizes the reconfigurable weight modules and summation module to update the winner-vector, as indicated in Fig. 4-10(b). The control unit run-time configures weight modules and summation module for this purpose, without interrupting the pipeline.

Therefore, the MRPA requires less circuit area when implementing the same function in comparison with the static pipeline. For instance, without reconfiguration, the area of the chip with 32-word parallelism will approximately increase 28% according to its layout in Fig. 4-11. The density efficiency of pipelined circuits is approximately proportional to c/f. Here, c represents the stage number of a static pipeline and f means the stage number of a reconfigurable pipeline, which realizes an equivalent function as in the c-stage static pipeline. The density efficiency of the pipeline part in the MRPA is 1.5 (c/f=6/4=1.5). The overhead of reconfiguration is due to the multiplexers, required in weight modules (N MUX2) and summation module ($2N$ MUX4). The area overhead of the additional multiplexers is 0.015

mm^2, occupying 0.7% of the chip area. The critical path results from EAM_N consisting of a multiplexer (MUX4) and a 46-bit full adder. The EAM_N causes the critical delay because of its 46-bit full adder. In all the other EAMs, the adders have shorter bit-length. To compare the density efficiency of MRPA with earlier studies, we use a comprehensive parameter-set consisting of normalized area, SRAM, and Matrix size (see Table 4-II).

4.5.1.2 Memory Utilization Efficiency

Memory utilization efficiency indicates the percentage of memory which can remain in constant use when number and dimensionality of weight vectors vary with applications. Managing allocation and mapping of memory is important to ensure the system can run a wide range of applications without modifications, which is particularly important to hardware where limited memory is available. An essential limitation of many hardware implementations for neural networks is the low memory utilization efficiency resulting from the fixed storage-space for each weight-vector. The dedicated SRAM-space for each weight-vector restricts the range of manageable weight-vector dimensionality and the number of individual SRAM-space units limits the weight-vector number. Any requests for a larger dimensionality than the size of the individual weight-vector SRAM-space or more weight vectors than the number of SRAM-space units cannot be fulfilled due to the fixed storage-space for each weight-vector. As a result, a certain amount of memory, which could have been utilized, is often wasted.

Without either emphasizing the necessity of on-chip memory for ASIC implementations or selecting FPGA with large internal SRAM blocks, we segregate the individual weight-memory from the PEs as a shared memory-pool so that the MRPA can support a flexible memory-space for each weight-vector. That is, the MRPA can sacrifice the vector dimensionality when more weight-vectors are needed or increase the vector dimensionality at the cost of fewer weight-vectors. During the compile (configuring) time (see Fig. 4-10), the MRPA performs memory allocation and memory mapping for the weight-vectors.

Shared memory-pool and partial-storage concept ensure that a large percentage of memory remains active, improving both the memory utilization efficiency and the application flexibility. The MRPA arranges an M-dimensional weight vector into P memory cells in each PSM. The separation signal for two weight vectors is asserted when the neuron address counter meets P. For example, a 3780-dimensional vector occupies 119 (N=32) cells in each PSM. A chip with 512 k-bit weight-memory and 32 word-parallelism (N=32) can process a range of configurations from 8 4096-dimensional weight-vectors to 1024 1-dimensional weight-vectors with 16-bit word-precision. Thus, both the vector dimensionality and number of weight-vectors

have a broad range of adjustability in this work.

Because of the internal fragmentation, the allocation concept applied in the design is more suitable for large-dimensional applications such as similarity searches in live video streams, DNA data, and so on. The memory utilization efficiency exceeds 50% as long as the vector-dimensionality is larger than *N/2*. For vectors whose dimensionality is not an integer multiple of *N*, the unused storage space is filled with "0", and thus lowers the memory utilization efficiency. The vectors need to be tailored to the memory sub-banks (Fig. 4-4), and an imperfect fit leads to wasted storage space corresponding to the unused portions of the memory sub-banks. Thus the design consumes more memory than its applications actually request, but largely increases the flexibility for usage in different applications. 1-dimensional weight vectors represent the worst-case situation, when only a portion of *1/N* of the resources available in one sub-bank are actually necessary. This means the worst-case situation leaves a large amount of the provided logic and memory unused. For weight vectors where the dimensionality is exactly an integer multiple of *N*, represents the best-case situation for the design, i.e., all sub-banks are completely filled. On the one hand, a small-size parallelism could mitigate the average resource-usage problem. However, the trade-off is that the average design performance will decrease and cost for implementing the communication between the reconfigurable modules will barely reduce for a small-size parallelism.

4.5.2 Post-Layout Results

For the purposes of proof-of-concept and prototyping, the architectural and algorithmic characteristics described above are verified by a 32 word-parallelism (N=32) test chip fabricated in 65 nm CMOS technology. The test chip, whose microphotograph and layout are shown in Fig. 4-11 with the principal parts highlighted, employs 32 parameterizable-storage modules, 32 weight modules, and 32 elementary-adder-modules in the summation module. The throughput of implementation is 512 bits per clock cycle with 16-bit word precision. The pipeline has 11 physical-stages, where the summation module occupies six physical-stages ($6=log_2N+1$ with N=32). Additionally, 512 k-bit SRAM is embedded for weight-vector storage and 97 k-bit SRAM is used for label memory (9 k-bit) and buffers (20 k-bit for input FIFO (first-in-first-out) buffers, 4 k-bit for output FIFO buffer, and 64 k-bit for input-vector SRAM). The input FIFO buffers consist of 16 k-bit (256 words × 64-bit) dual-port SRAM with independent clock for input vector, and 4 k-bit (512 word×8-bit) SRAM for input label. The number of input-vectors stored in the input SRAM relies on the vector dimensionality. For

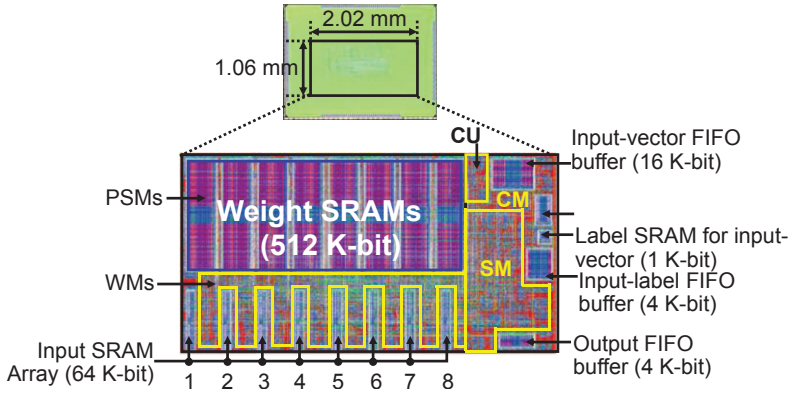


Fig. 4-21. Micrograph and layout of the prototype chip in 65 nm CMOS technology.

example, the input SRAM can store 128 32-dimensional input vectors or one 4096-dimensional input vector. The read bandwidth of the input SRAM can reach 76.8 Gbit/s at 150 MHz working frequency. Nevertheless, the write bandwidth requires only 76.8/($NN\times P$) Gbit/s since each input vector has to be processed $NN\times P$ times. Accordingly, the designed write bandwidth of the input buffer with 16 Gbit/s can satisfy the requirements of the input SRAM when $NN\times P$ exceeds five

Due to the pipeline reconfiguration and the modularity methodology, the prototype chip achieves high density efficiency and memory utilization efficiency with a core area of 2.14 mm^2, and an average power consumption of 9.4 mW at 100 MHz and 0.8 V supply voltage. The embedded 609 k-bit memory occupies 41% of the chip area. Normally, the power consumption is directly proportional to this area. Fig. 4-12 shows the measured total energy per operation and the maximum working frequency of the prototype. When the voltage is lower than 0.6V, functional failures occur because the failure probability of SRAM cells significantly increases at low nominal voltages near the transistor threshold.

The well-established performance metrics MCPS (Million Connections per Second) and MCUPS (Million Connection Updates per Second) are separately used to evaluate the chip

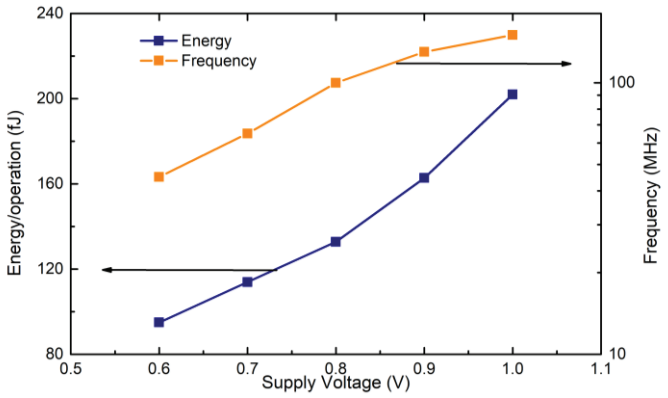


Fig. 4-23. Measured energy per operation and maximum working frequency of the test chip.

when operating in recognition mode or learning mode, respectively. The MCPS and MCUPS metrics are defined in eqs. (4-7) and (4-8), where *F*, *NN*, and *M* are the working frequency, the number of weight-vectors, and vector dimensionality, respectively. $C_{recognition}$ in eq. (4-9) is the number of required clock-cycles to recognize each input and $C_{learning}$ in eq. (4-10) is the number of required clock-cycles to update the winner vector. In eq. (4-9), apart from the summation module with $log_2 N + 1$ physical-stages, parameterizable-storage modules, weight modules, and comparison module contribute to the five physical-stages. The first weight-vector needs $(P - 1)$ additional-clocks because of the partial-calculation and accumulation to determine the squared Euclidean distance. The remaining (*NN*-1) weight-vectors require $P \times (NN - 1)$ clock-cycles, which results from the pipelined and partial processing. As for the learning mode, the MRPA subsequently shifts to the adaption phase after finishing the nearest-neighbor-search phase. The configuration of weight modules and summation module for the adaption is shown in Fig. 4-6. The pipeline structure during the adaption phase is depicted in Fig. 4-10(b). Operation of parameterizable-storage modules, weight modules, summation module, and writing of the intermediate calculation results back to the parameterizable-storage modules cause five clocks, as illustrated in eq. (4-10). The processing of the remaining (*P-1*) *N*-dimensional components of the winner requires (*P-1*) clocks.

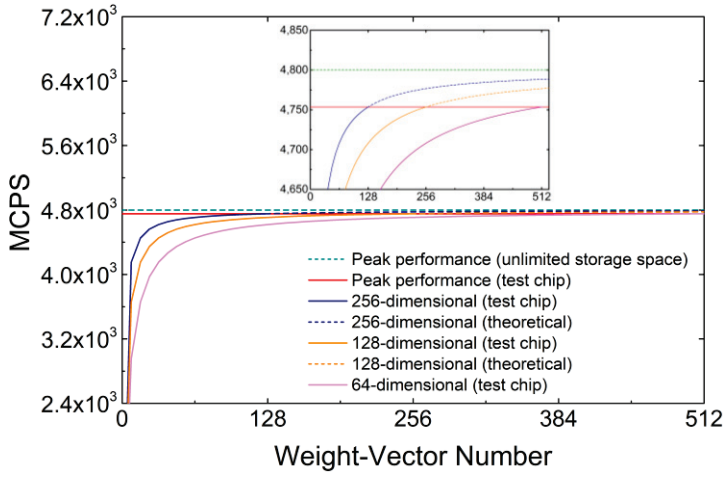
$$MCPS = \frac{F \times NN \times M}{C_{recognition}} \tag{4-7}$$

$$MCUPS = \frac{F \times NN \times M}{C_{learning}} \tag{4-8}$$

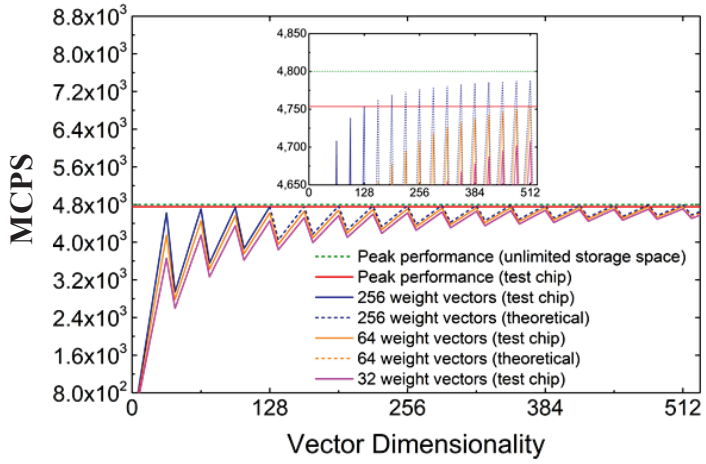
$$\begin{aligned} C_{recognition} &= 3 + (log_2 N + 1) + (P - 1) + 2 + P \times (NN - 1) \\ &= 5 + log_2 N + P \times NN \end{aligned} \tag{4-9}$$

$$\begin{aligned} C_{learning} &= C_{recognition} + 5 + (P - 1) \\ &= 5 + log_2 N + P \times NN + 5 + (P - 1) \\ &= 9 + log_2 N + P \times NN + P \end{aligned} \tag{4-10}$$

The performances of the prototype chip with $N = 32$ word-parallelism for recognition in terms of MCPS and learning in terms of MCUPS are illustrated in Fig. 4-13 and Fig. 4-14, respectively. These diagrams show the dependency between the performance metrics and the number of weight vectors, (Fig. 4-13 (a), Fig. 4-14 (a)), as well as the vector dimensionality, (Fig. 4-13 (b), Fig. 4-14 (b)). The shared memory-pool allows the MRPA to extend the number of weight vector at the cost of lower vector dimensionality or conversely.



(a)

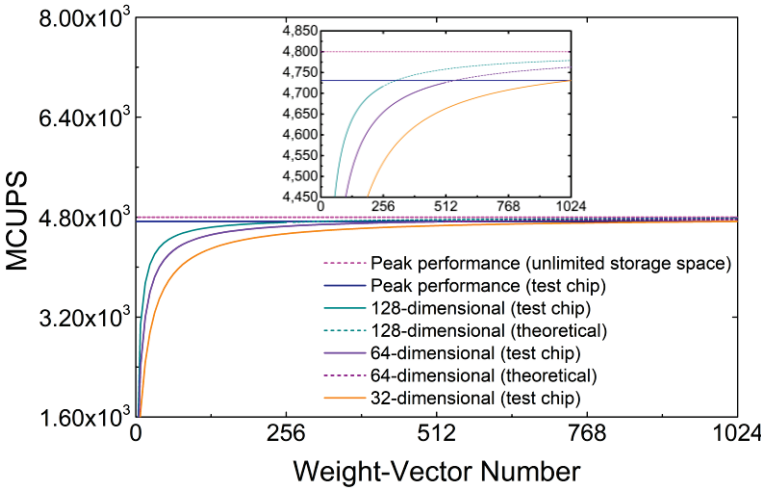


(b)

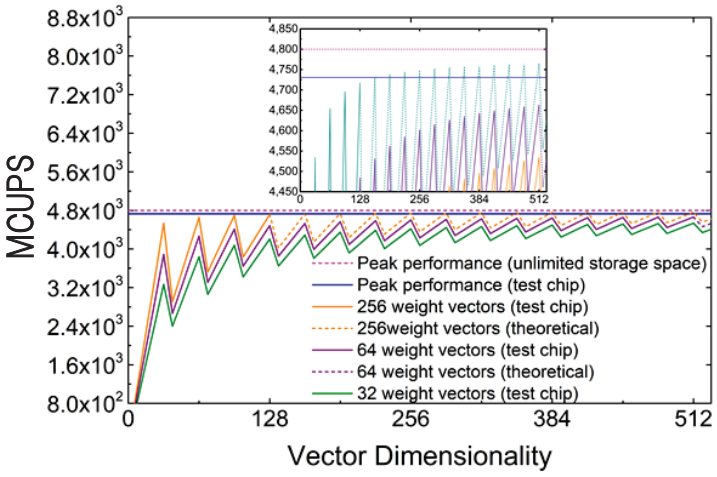
Fig. 4-25. Prototype performance in MCPS (Million Connections per Second) as a function of the number of weight vectors (a) and vector dimensionality (b).

The prototype attains the maximum MCPS performance under the condition that the entire circuit is working. At peak performance, M can be exactly divided by N without a remainder. For the test chip peak performance is obtained when M is a multiple of 32 (N=32). Meanwhile, the number of weight vectors NN should be the largest that can be accommodated in the on-chip SRAM for the weight-vectors (512 k-bit). Accordingly, the prototype reaches its peak performance when $NN \times P \times N = 32768$ (32768=32×1024). The requirement for dimensionality results from the partial-storage concept, which ensures the whole specific function modules are efficiently working. As for MCUPS, only one condition can result in the maximum performance, in which the vector dimensionality should be exactly equal to the word-parallelism (32), and NN should have the largest possible value (1024). This is because when M is equal to N (P=1), the adaption time is the minimum.

According to the equations (4-7) to (4-10), the MCPS and MCUPS performances can be



(a)



(b)

Fig. 4-27. Prototype performance in MCUPS (Million Connection Updates per Second) as a function of the number of weight vectors (a) and vector dimensionality (b).

improved by increasing the word-parallelism N (which results in reduction of P) and working frequency F. The worst case of the performance of the prototype chip with 32 word-parallelism and 150 MHz working frequency is 25 MCPS and 9.5 MCUPS when one 1-dimensional weight vector is store as the weight vector. On the other hand, the best case with 4753.578 MCPS and 4730.703 MCUPS occus when M is divisible by N and the memory blocks for weight vectors are fully used. In the fabricated prototype, N=32 and 1024 words for each weight-vector memory block are chosen, so that best performance is achieve e.g. in the case when $NN \times M$=1024×32. As an ideal case, if unlimited storage-space is provided, the maximum MCPS and MCUPS would be 4800. The theoretical results, when a chip can provide unlimited memory space for weight vectors, are also included in Fig. 4-13 and Fig. 4-14. However, the differences between the theoretical best performance and achieved performance are so small that the curves almost overlap. In other words, if the word-parallelism is fixed, increasing the

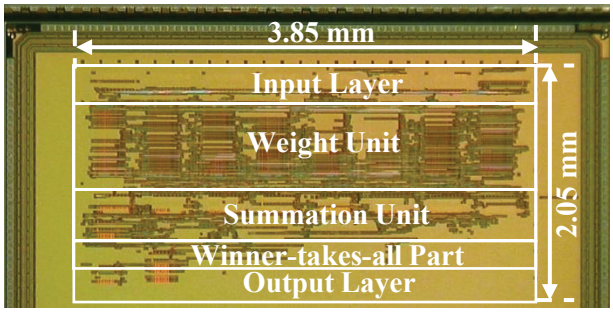


Fig. 4-29. Micrograph of the fabricated chip in 180 nm CMOS technology with 8-word parallelism for the PPPI architecture.

memory-space will only slightly improve the performance at the cost of a quite large Si-area. The performance tends to saturate when either the number of weight vectors *NN* or the vector dimensionality *M* is large. This saturation occurs because both cases just improve the utilization efficiency of the circuits whereas the word-parallelism of the prototype remains unchangeable at 32. The saw-tooth shape of the performance dependency in Fig. 4-13 (b), Fig. 4-14 (b) is caused by the partial processing of the vectors.

The prototype of the LVQ VLSI realization based on the R-PPPI architecture (p=8) was fabricated in 180 nm CMOS technology as shown in the photomicrograph of Fig. 4-15. Since 8-word parallelism and 16 bit precision are chosen in this design, the R-PPPI architecture has a throughput of 128 bits per clock cycle and a pipeline latency of 8 stages. Moreover, the designed LVQ ASIC with on-chip learning and classification, which has core area of 7.89 mm^2, can handle at maximum 4096-dimensional vectors. In the classification mode, except for 8 clock cycles (106 ns at 75 MHz) of the pipeline latency, each d-dimensional ($d \leq 4096$) vector can be processed in every $\lceil d/8 \rceil - 1$ clock cycles. Consequently, a large number of different applications can be handled due to the high flexibility in vector dimensionality and the reference-vector number. In principle, the designed LVQ on-chip learning and recognition hardware can accommodate any application with feature vectors of up to 4096 dimensions. For example, in the case of 3780-dimensional feature vectors (Histogram of Gradient (HOG) feature [29] used in pedestrian detection), the partial storage parameter m ($=\lceil d/p \rceil$) is defined as 473, where the unused 4 words in the last partial group of components are simply filled up with zeros. In this way, each test 3780-d feature vector can be classified in 473×R clock cycles where R, usually below 100, is the reference number (6.3R μs at 75 MHz). Furthermore, the on-chip learning with very high learning speed enables the application in online machine learning.

4.5.3 Architecture and Algorithmic Optimization Results

A comparison with previous state-of-the-art works is illustrated in Table 4-II. To make a fair comparison, we normalized the area and power consumption to the 65 nm CMOS technology by using the "constant field scaling theory" [54]. The developed MRPA not only ensures high

Table 4-III Comparison Results between previous works and our work

		[46]	[38]	[39]	[40]	[41]	[42]	[43]	Our work
CMOS technology		180 nm	90 nm (FPGA)	90 nm (FPGA)	220 nm (FPGA)	220 nm (FPGA)	90 nm (FPGA)	- FPGA	65 nm
Algorithm		Leanring :LVQ	LVQ	LVQ	LVQ	LVQ	LVQ	LVQ	LVQ
Architectural features	Neurons	parallel	sequentially	parallel	parallel	parallel	parallel	parallel	sequentially
	Vector component	parallel	sequentially	sequentially	sequentially	sequentially	parallel	sequentially	partially-parallel
Normalized area[a, b, c]		5.84 mm^2	-	-	-	-	-	-	2.14 mm^2
SRAM (k-bit)[a]		256	576	58.23	49	35.93 (weight vectors)	56.47	34	609
Matrix size[a]	Vector size (Dimension)	1-64(12 bit)	23 (16 bit)	21 (14 bit)	23 (8 bit)	23 (8 bit)	23 (8 bit)	64 (16 bit)	1-4096 (16 bit)
	Weight-vector number	256	350	12	49	25	25	32	1-1024
Memory utilization efficiency[e]		75%	21.8%	5.9%	18.0%	12.5%	8.0%	94.1%	84%
Normalized power (mW)[b, d]		82.15 (50 MHz @1.8 V)	88.67 (50 MHz)	597.76 (50 MHz)	1233.63 (100 MHz)	-	-	-	9.4 (100 MHz @0.8V) 21.5 (150 MHz @1V)
Processor reconfigurabilty		Between PE array processor and SOM network	No	No	No	No	No	No	Between learning and recognition mode

[a]Normalized area, SRAM, and Matrix size comprise a comprehensive parameter-set to evaluate density efficiency.

[b]We normalized the area and power consumption to the 65-nm technology according to the constant field scaling theory [54]. For the area, we also considered the influences of numerical precision because the precision directly affects the area. The power consumption is known to reduce less than predicted by the scaling theory, when the gate length becomes shorter than 100 nm [55]. Therefore, the predictions for previous works are best-case values for these designs. The actual performances are far worse than the predictions. In addition to the gate length, the working frequency, voltage, and the embedded SRAM influence the power consumption significantly.

[c]Normalized area (*NArea*) is the core area normalized to the 65-nm technology and 16-bit precision given by eq. (4-11). The SRAM volume also greatly affects the normalized area.

$$NArea = \frac{Core\ area}{(CMOS\ tech./65)^2 \times (Precision/16)}\ [mm^2] \quad (4\text{-}11)$$

[d]Normalized power (*NPw*) is the power consumption normalized to the 65-nm technology given by eq. (4-12).

$$NPw = \frac{Power\ cons.}{(CMOS\ tech./65)^2}\ [mW] \quad (4\text{-}12)$$

[e]Memory utilization efficiency (*MUE*) refers to the maximum number of weight-vector with the maximum dimensionality that the SRAM can handle.

$$MUE = \frac{(max.\ number\ of\ weight\ vectors) * (weight\ vector\ dimensionality)}{SRAM/bit\ precision} \quad (4\text{-}13)$$

performance and flexibility, but also efficiently improves the area and power efficiency. The MRPA offers more flexibility than the non-reconfigurable ASIC because of its module reuse and partial processing. No dedicated learning or recognition circuits are needed in MRPA, while a non-reconfigurable ASIC requires specific hardware for each sub-function. Because of the module reuse, the prototype requires less circuit area. The ALUs, such as multipliers, adders, and subtractors, which occupy large Si-area, serve in multiple sub-functions. In particular, not only do the learning and recognition modes share the same hardware, but the nearest-neighbor-search phase also uses these same modules in common with the adapting phase. The partial processing provides flexible vector size and weight-vector number as shown in Table 4-II with good memory utilization efficiency. The MRPA can process a range of scales from 1 to 4096-dimensional weight-vectors. The unfixed memory-space for each weight-vector contributes to the large vector capacity and high memory utilization efficiency.

Table 4-III compares the MCPS and MCUPS performances, and the connection energy achieved in this work with the results of well-known previous research. With respect to performance, the MRPA outperforms the non-reconfigurable ASIC in [45] and FPGA implementations [37-42], while the GPU performs the best, consuming considerable connection energy. We applied two circuit-level techniques to the MRPA for the purpose of improving the implementation efficiency. The first one is the hardware sharing which reduces the hardware consumption for computation, and the second one is the pipeline architecture which accelerates the computing speed. The *P N*-dimensional partial-vectors share the *N* weight modules and elementary-adder-modules instead of having *P*N* independent weight modules and elementary-adder-modules. The analysis of hardware consumption for computation consists of two parts: the distance calculation during the nearest-neighbor-search phase, and the winner adaption during the adaption phase. The multiplication is the most complex computation in the LVQ algorithms. No matter which distance metric is used to calculate the distances, the multiplication is necessary because of the winner adaption. A straightforward realization of either the distance calculation or the winner adaption tends to increase a large hardware overhead, which will obviously prevent the achievement of high throughput with reasonable silicon area and power consumption. For example, for the fully-parallel SIMD-implementation with a capacity of *NN M*-dimensional vectors, the Manhattan distance (*MD*) needs *NN*M* subtractors and *NN*(M-1)* adders, and the winner adaption requires *NN*M* subtractors, *NN*M* adders and *NN*M* multipliers. In total, the fully-parallel SIMD-methods need *NN*M* subtractors, *NN*M* adders and *NN*M* multipliers. The non-fully-parallel SIMD implementation reduces the hardware consumption for computation to either

Table 4-V PERFORMANCE COMPARISONS FOR LEARNING AND RECOGNITION

	Distance metrics	Hardware consumption	*NN*	*M*	*F* (MHz)	MCUPS	MCPS	CEL[a]	CER[a]
[34]	ED^2	-	3755	160	2530	54731.3	-	14769.5	-
[46]	*MD*	*NN*M*	256	32	50	-	258	-	318.41
[38]	ED^2	*1*	350	23	50	-	23.95	-	3702
[39]	ED^2	*NN*	12	21	50	11.29	136.95	52940	5360
[40]	ED^2	*NN*	49	23	100	1115.84	1543.83	1105.56	799.07
[41]	*MD*	*NN*	25	23	25	-	625	-	-
[42]	*ED*	*NN*M*	25	23	25	6.25	-	-	-
[43]	*MD*	*NN*	32	64	25	-	6.9	-	-
Our work	ED^2	*N*	256	128	150	4730.703	4753.578	4.54	4.52

MD: Manhattan distance. ED^2: Squared Euclidean distance. *ED*: Euclidean distance.
[a]Connection energy (CE) is the energy per connection, which can reflect the fundamental efficiency of the circuit and is invariant to performance changes [47, 49]. The CE for learning (CEL) and recognition (CER) are shown in eq. (4-11) and eq. (4-12).

$$CEL = \frac{Energy}{Connenction\ number\ for\ update} = \frac{NPw}{Connenction\ number\ for\ update/second} = \frac{NPw}{MCUPS} \times 10^3[pJ] \quad (4\text{-}11)$$

$$CER = \frac{Energy}{Connenction\ number} = \frac{NPw}{Connenction\ number/second} = \frac{NPw}{MCPS} \times 10^3[pJ] \quad (4\text{-}12)$$

NN or *M* for effective processing [56]. Furthermore, the node parallelism (weight-vector or neuron parallelism) is the most used and maybe the most natural mapping for SIMD computers [57]. The normal case is that a non-fully-parallel SIMD processes *NN* parallel weight vectors. The MRPA decreases the hardware consumption for computation from *NN*M* to *N* by following the multiple-instruction-multiple-data (MIMD) processing manner, which is partially parallel and pipelined. The *M*-dimensional vector is divided into *P N*-dimensional vectors, which are processed in the pipeline. As a result, both the squared Euclidean distance calculation and the winner adaption require *N* subtractors, *N* adders, and *N* multipliers. The MRPA reuses the ALUs for NNS and adaption phases. Altogether the MRPA consumes *N* subtractors, *N* adders, and *N* multipliers. Therefore, the hardware consumption for computation of MRPA is increasing linearly with *N* for the LVQ algorithm. In addition, the MRPA reduces the computational complexity, while providing appropriate results for learning and recognition, by selecting the squared Euclidean distance rather than the Euclidean distance or the Manhattan distance. In real-world applications, the squared Euclidean distance and the Euclidean distance often provide higher accuracy in distance comparisons than the Manhattan distance [58]. Because we only need to compare the distances values, the squared Euclidean distance without root operation is the most efficient option. Although the hardware sharing strategy greatly

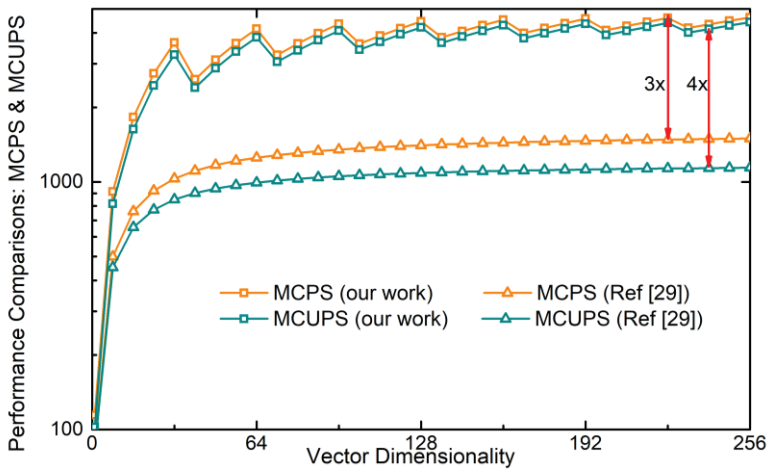


Fig. 4-31. Performance comparisons between our work and the neuron parallelism solution [40] in learning and recognition modes, when working frequency and weight-vector number are same.

facilitates complexity reduction, it still leaves room for improvements. A possible method is to replace the ripple-carry adder in elementary-adder-module with a carry-look-ahead adder, in order to reduce the computation time and to increase the clock frequency for the pipeline.

The power dissipation can be defined as (connection/second) ×(energy/connection). The connection energy, which reflects the energy per connection, represents the efficiency of the LVQ hardware [47, 59]. Our work has very good performance in terms of the connection energy for learning and recognition, as verified in Table 4-III. Unfortunately, the comparison remains incomplete because not all FPGA implementations provide detailed results for the power consumption.

In summary, no single architecture performs best in terms of performance and energy efficiency for the complete application space. The neuron parallel approach can effectively handle the applications with a high number of neurons and lower dimensionality. The applications with high-dimensional vectors can benefit from the MRPA.

Fig. 4-16 shows that, given the same working frequency and 32 weight-vectors, our design can achieve a performance improvement of approximately 3x for MCPS and 4x for MCUPS in comparison to the neuron (weight vector) parallelism solution in [40].

As for the accuracy loss of the hardware implementation, it mainly resulted from the truncation operation of the fixed-point operation. We carried out a comparison of fixed-point with floating-point operations to benchmark the impact of the round-off errors. For this purpose the SOM-based image compression was used, which is a lossy data compression method [47] and covers the learning process for generating the codebook of an image encoder and the recognition process for encoded images. Here, the peak signal-to-noise ratio (PSNR) is an

appropriate metric for the evaluation of the round-off errors. We have simulated the round-off error using various learning rates (from 0.07 to 0.9) and vector dimensionalities (non-overlapped pixel-block sizes for image compression) regarding PSNR. The used number of learning iterations and weight vectors was 256 and 30, respectively. The round-off error of MRPA mainly resulted from the truncation operation before writing the adapted winner data into the parameterizable-storage modules. As shown in Fig. 4-17, where the *x* axis specifies the learning rate and the *y* axis denotes the PSNR, the fixed-point number with 16-bit precision leads only to a very small PSNR loss of 0.128 dB, because the MRPA preserves the precision as much as possible and no overflow instances occur in the internal signals. These findings justify the choice of fixed-point operators, which are much more hardware-friendly than the floating-point operators.

4.6 Summary

In this chapter, a hardware architecture called MRPA (a Modular and Reconfigurable Pipeline Architecture), realizing learning vector quantization (LVQ) and taking advantage of pipeline reconfiguration and modularity, was developed and verified by a prototype chip in 65-nm CMOS technology. The pipeline reconfiguration leads to a reduction in computation time and high efficiency of integration density. The modularity contributes to easy scalability in both upward- and downward-compatible fashion. Additionally, the shared memory-pool increases the flexibility for both the dimensionality and the number of weight vectors. Further, the implemented parameterization adds flexibility to the choice of adaption strategies. Consequently, the designed VLSI prototype implementation in this study attains high performance regarding MCPS and MCUPS, optimum Si-area efficiency, and verifies an enhancement of the usability for embedded artificial-intelligence applications. Before

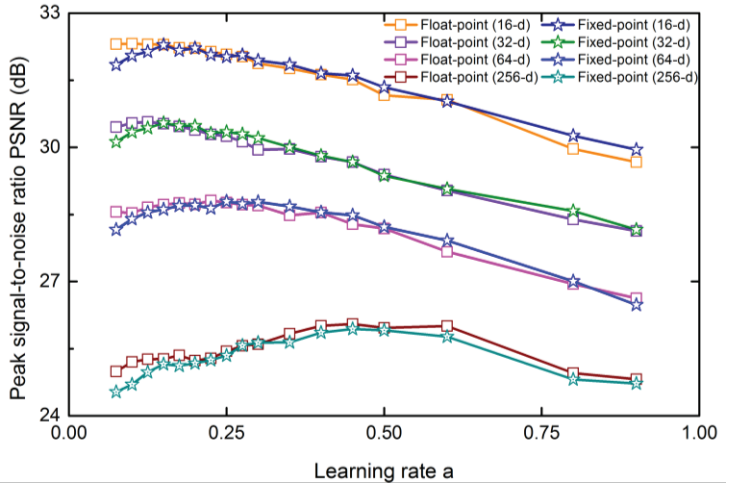


Fig. 4-33. PSNR comparison of float-point operators with fixed-point operators.

developing the MRPA, a memory-based VLSI realization for LVQ neural networks using the R-PPPI architecture was designed for on-chip learning and classification and fabricated in 180 nm CMOS technology. The short learning time and high flexibility improves the applicability for a large number of practical applications. The R-PPPI architecture is verified to execute the dual modes of learning and recognition with very low power dissipation and small Si-area consumption. Moreover, the nearest neighbor search, the part with the highest computational demand, is the critical computational complexity solved by this reconfigurable R-PPPI architecture as well. The fabricated chip has furthermore demonstrated the high learning and classification speed.

References

[1] Botros N M, Abdul-Aziz M. Hardware implementation of an artificial neural network using field programmable gate arrays (FPGA's)[J]. *IEEE Transactions on Industrial Electronics*, 41(6): 67-665, 1994.

[2] Hendry D C, Duncan A, Lightowle N. IP core implementation of a self-organizing neural network[J]. *IEEE Transactions on Neural Networks,* 2003, 14(5): 1085-1096.

[3] Blake J J, Maguire L P, McGinnity T M, et al. The implementation of fuzzy systems, neural networks and fuzzy neural networks using FPGAs[J]. *Information Sciences*, 1998, 112(1): 151-168.

[4] Izeboudjen N, Farah A, Titri S, et al. Digital implementation of artificial neural networks: from VHDL description to FPGA implementation[M]//*Engineering Applications of Bio-Inspired Artificial Neural Networks*. Springer Berlin Heidelberg, 1999: 139-148.

[5] Anguita D, Boni A. Improved neural network for SVM learning[J]. *IEEE Transactions on Neural Networks*, 2002, 13(5): 1243-1244.

[6] Liu J, Brooke M, Hirotsu K. A CMOS feedforward neural-network chip with on-chip parallel learning for oscillation cancellation[J]. *IEEE Transactions on Neural Networks*, 2002, 13(5): 1178-1186.

[7] M. Franzmeier, C. Pohl, M. Porrmann, and U. Rückert, Parallel Computing in Electrical Engineering, 2004. PARELEC 2004. *International Conference on*, 309 (2004).N. Sudha, Real-Time Imaging 10.1, 31 (2004).

[8] Sudha N. An ASIC implementation of Kohonen's map based colour image compression[J]. *Real-Time Imaging*, 2004, 10(1): 31-39.

[9] Rajah A, Hani M K. ASIC design of a Kohonen neural network microchip[C]//Semiconductor Electronics, 2004. ICSE 2004. *IEEE International Conference on*. IEEE, 2004: 4 pp.

[10] Arena P, Fortuna L, Frasca M, et al. A CNN-based chip for robot locomotion control[J], *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2005, 52(9): 1862-1871.

[11] Gorgoń M, Wrzesiński M. Neural network implementation in reprogrammable FPGA devices–an example for MLP[M]//*Artificial Intelligence and Soft Computing–ICAISC* 2006. Springer Berlin Heidelberg, 2006: 19-28.

[12] Li H, Zhang D, Foo S Y. A stochastic digital implementation of a neural network controller for small wind turbine systems[J]. *IEEE Transactions on Power Electronics*, 2006, 21(5): 1502-1507.

[13] Koickal T J, Hamilton A, Tan S L, et al. Analog VLSI circuit implementation of an adaptive neuromorphic olfaction chip[J]. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2007, 54(1): 60-73.

[14] Kyrkou C, Theocharides T. A parallel hardware architecture for real-time object detection with support vector machines[J]. *IEEE Transactions on Computers*, 2012, 61(6): 831-842.

[15] J. Bailey and D. Hammerstrom, “Why VLSI implementations of Associative VLCNs require connection multiplexing?” in *IEEE International Conference On Neural Network*, USA, 1998, vol. 2, pp. 173–180.

[16] J. Zhu, and P. Sutton, “FPGA implementations of neural networks—a survey of a decade of progress.” in *Proc. Conf. Field Programm. Logic*, Portugal, Lisbon, 2003, pp. 1062–1066.

[17] P. Knag, J. K. Kim, T. Chen, and Z. Zhang, “A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 4, pp. 1070–1079, Apr. 2015.

[18] P., Mazumder, D. Hu, I. Ebong, X. Zhang, Z. Xu and S. Ferrari, “Digital implementation of a virtual insect trained by spike-timing dependent plasticity,” *Integration the VLSI Journal*, vol. *54*, pp. 109–117, Jun. 2016.

[19] U. Ramacher, “SYNAPSE—A neurocomputer that synthesizes neural algorithms on a parallel systolic engine,” *Journal of Parallel and Distributed Computing*, vol. 14, no. 3, pp. 306–318. Mar. 1992.

[20] Willshaw D J, Von Der Malsburg C. How patterned neural connections can be set up by self-organization[J]. Proceedings of the Royal Society of London B: Biological Sciences, 1976, 194(1117): 431-445.

[21] Kohonen T. Self-organized formation of topologically correct feature maps[J]. Biological cybernetics, 1982, 43(1): 59-69.

[22] Fang X, Thole P, Göppert J, et al. A hardware supported system for a special online application of self-organizing map[C]//Neural Networks, 1996., IEEE International Conference on. IEEE, 1996, 2: 956-961.

[23] Kaski S. Data exploration using self-organizing maps[C]//ACTA POLYTECHNICA SCANDINAVICA: MATHEMATICS, COMPUTING AND MANAGEMENT IN ENGINEERING SERIES NO. 82. 1997.

[24] Céréghino R, Park Y S. Review of the self-organizing map (SOM) approach in water resources: commentary[J]. Environmental Modelling & Software, 2009, 24(8): 945-947.

[25] Kohonen T. Improved versions of learning vector quantization[C]//Neural Networks, 1990., 1990 IJCNN International Joint Conference on. IEEE, 1990: 545-550.

[26] M. Boubaker, M. Akil, K. Ben Khalifa, T. Grandpierre, and M.H. Bedoui, “Implementation of an LVQ neural network with a variable size: algorithmic specification, architectural exploration and optimized implementation on FPGA devices,” Neural Computing and Applications, vol. 19, no. 2, pp. 283-297, 2010.

[27] T. Kohonen, Self-Organizing Maps, Springer-Verlag, Berlin, 1995, (Second extended edition 1997).

[28] F. An, T. Akazawa, S. Yamasaki, L. Chen, and H. J. Mattausch, "VLSI realization of learning vector quantization with hardware/software co-design for different applications," Japanese Journal of Applied Physics, vol.54, no.4s, pp. 04DE05, 2015.

[29] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” Proc. of IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 886-893. 2005.

[30] N. Dalal, INRIA Human Dataset (2005) [http://pascal.inrialpes.fr/data/human/].

[31] P. Kolinummi, P. Pulkkinen, T. Hämäläinen, and J. Saarinen, “Parallel implementation of self-organizing map on the partial tree shape neurocomputer,” *Neural processing letters*, vol. 12, no. 2, pp. 171-182, Oct. 2000.

[32] E. M. Imah, and R. Sulaiman, “Online Kernel AMGLVQ for Arrhythmia Hearbeats Classification,” *Kursor*, vol. 8, no. 4, pp.159-168, Dec. 2017.

[33] M. Abadi, M, S. Jovanovic, K. B. Khalifa, S. Weber and M. H. Bedoui, “A Scalable Flexible SOM NoC-Based Hardware Architecture,” in Proceedings of the 11th International Workshop WSOM, USA, Texas, 2016, pp. 165–175.

[34] T. Su, S. Li, P. Ma, S. Deng, and G. Liang, “Scalable Prototype Learning Using GPUs,” In International Conference Image Analysis and Recognition (ICIAR), Portugal, Algarve, Vilamoura, 2014, pp. 309-319.

[35] J. Yang, Y. Yang, Z. Chen, L. Liu, J. Liu and N. Wu, “A Heterogeneous Parallel Processor for High-Speed Vision Chip,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. PP, no. PP, pp. PP, 2016.

[36] J. Kim and P. Mazumder, “Energy-Efficient Hardware Architecture of Self-Organizing Map (SOM) for ECG Clustering in 65nm CMOS,” *IEEE Transactions Circuits and Systems II: Express Briefs*, vol. PP, no. PP, pp. PP, 2017.

[37] X. Zhang, F. An, L. Chen and H. J. Mattausch, “Reconfigurable VLSI implementation for learning vector quantization with on-chip learning circuit,” *Japanese Journal of Applied Physics*, vol. 55, no. 4S, pp. 04EF02, Mar. 2016.

[38] C. Chalbi, and M. H. Bedoui, “Implementation of a low-power LVQ architecture on FPGA,” *IET Circuits, Devices & Systems*, vol. 11, no. 6, pp. 597-604, May 2017.

[39] A. G. Blaiech, K. B. Khalifa, M. Boubaker, and M. H. Bedoui, “LVQ neural network optimized implementation on FPGA devices with multiple-wordlength operations for real-time systems,” *Neural Computing and Applications*, pp.1-20, Jul. 2016.

[40] N. Chalbi, K. B. Khalifa, M. Boubaker, and M. H. Bedoui, “Implementation of a novel LVQ neural network architecture on FPGA,” *International Journal of Artificial Intelligence and Soft Computing*, vol. 2, no. 3, pp.163-173, Jan. 2010.

[41] M. Boubaker, K. B. Khalifa, B. Girau, M. Dogui, and M. H. Bedoui, “On-line arithmetic based reprogrammable hardware implementation of LVQ neural network for alertness classification,” *IJCSNS International Journal of Computer Science and Network Security*, vol. 8, no. 3, pp.260-266, Mar. 2008.

[42] M. Boubaker, M. Akil, K. B. Khalifa, T. Grandpierre and M. H. Bedoui, “Implementation of an LVQ neural network with a variable size: algorithmic specification, architectural exploration and optimized implementation on FPGA devices,” *Neural Computing and Applications*, vol. 19, no. 2, pp. 283-297, Mar. 2010.

[43] M. Kugler, and H. S. Lopes, “A configware approach for the implementation of a LVQ neural network,” *International Journal of Computational Intelligence Research*, vol. 3, no. 1, pp. 21-25, Jan. 2007.

[44] K. Tanaka, “Hardware Design of Embedded Systems for Security Applications,” in *Embedded Systems-High Performance Systems, Applications and Projects*, 1st ed., Croatia, 2012, pp. 235–237.

[45] C. Plessl and M. Platzner, “Virtualization of Hardware-Introduction and Survey,” in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, USA, Las Vegas, Nevada, 2004, pp. 63–69.

[46] C. Shi, J. Yang, Y. Han, Z. Cao, Q. Qin, L. Liu, N. Wu, and Wang, “A 1000 fps Vision Chip Based on a Dynamically Reconfigurable Hybrid Architecture Comprising a PE Array Processor and Self-Organizing Map Neural Network,” *IEEE Journal of Solid-State Circuits*, vol. 49, no. 9, pp. 2067–2082, Sep. 2014.

[47] J. Lachmair, T. Mieth, R. Griessl, J. Hagemeyer and M. Porrmann, “From CPU to FPGA-Acceleration of self-organizing maps for data mining,” in *IEEE International Joint Conference on Neural Networks (IJCNN)*, USA, Alaska, 2017, pp. 4299-4308.

[48] J. Max, “Quantizing for Minimum distortion,” *IRE Trans. Inform. Theory*, vol. IT-6, no. 2, pp. 7-12, Mar. 1960.

[49] T. Kohonen, “Improved version of learning vector quantization”, *International Joint Conference on Neural Networks*, pp. 545-550, 1990.

[50] P. Lysaght und J. Dunlop, “Dynamic Reconfiguration of FPGAs”, Selected papers from the Oxford *1993 international workshop on field programmable logic and applications on More FPGAs*, pp. 82-94, 1994.

[51] F. An, X. Zhang, L. Chen, H.J. Mattausch, "A Memory-Based Modular Architecture for SOM and LVQ with Dynamic Configuration", *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, pp. 234-241, 2016.

[52] S. A. Guccione and M. J. Gonzalez, “Classification and performance of reconfigurable architectures.” in *Lecture Notes in Computer Science*, Berlin, Heidelberg, Springer Berlin Heidelberg, 1995, ch. 46, vol. 975, pp. 439–448.

[53] P. Frasconi, M. Gori and G. Soda, “Links between LVQ and backpropagation,” *Pattern Recognition Letters*, vol. 18, no. 4, pp. 303-310, Apr. 1997.

[54] R. H.Dennard, F. H.Gaensslen, L. Kuhn and H. N. Yu, “Design of micron MOS switching devices,” in *Proceeding of IEEE 1972 International Electron Devices Meeting*, USA, Washington, DC, 1972, pp. 168–170.

[55] P. A. Salvadeo, Á. C. Veca and R. C. López, “Historic behavior of the electronic technology: The Wave of Makimoto and Moore's Law in the Transistor’s Age,” in *Proceeding of VIII Southern Conference on Programmable Logic*, Brazil, Bento Gonçalves, 2012, pp. 1–5.

[56] H. Inoue, T. Moriyama, H. Komatsu and T. Nakatani, “AA-sort: A new parallel sorting algorithm for multi-core SIMD processors,” in *Proceedings of the 16th International*

Conference on Parallel Architecture and Compilation Techniques, USA, Washington, DC 2007, pp. 189-198.

[57] T. Nordström, "Designing parallel computers for self organizing maps," in *Proceedings of the 4th Swedish Workshop on Computer System Architecture (DSA-92)*, Sweden, Linkoping, 1992, pp. 13-15.

[58] S. Salzberg, "Distance metrics for instance-based learning," *Methodologies for Intelligent Systems*, vol. 542, pp. 399-408, 1991.

[59] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," *ACM SIGARCH Computer Architecture News*, vol. 38, No. 3, pp. 37-47, June 2010.

CHAPTER 5: Conclusions and Future Directions

5.1 Summary of Contributions

5.1.1 Hardware-Oriented Algorithm Design

In this study, the original HOG algorithm is implemented in an improved hardware-oriented way, applying a cell-based scan manner. Different from the state-of-the-art hardware implementations, without block-based normalization, the cell-based HOG descriptor extraction units apply a pixel-based pipelined architecture that can synchronize to the working frequency of the image sensor, thus enabling the flexibility of input-image sizes and detection in scaled images. Subsequent partial recognition for all overlapped windows to which each cell belongs uses the same processing manner for the pixels from the image sensor as in the cell-based sliding window paradigm for recognition.

In addition, my developed modular and reconfigurable pipeline-architecture (MRPA) for LVQ neural networks has the following advantages. First, the MRPA accelerates the computational speed and provides high integration density by the implementation of pipeline reconfiguration. All the weight-vectors share the same arithmetic and logic units (ALUs), rather than having individual ALUs. Meanwhile, the MRPA improves the memory-utilization efficiency by segregating the weight-memory blocks from the processing elements (PEs) as a shared memory pool. The memory sharing scheme also increases the flexibility of the weight-vector, in contrast to the SIMD methods, which directly map neurons to PEs. The size of an individual weight-memory block and the number of weight-memory blocks limit the range of manageable dimensionality and number of weight-vectors. The MRPA overcomes this limitation. Both the dimensionality and number of weight-vectors are adaptable to a wide range of applications. Moreover, the modularity of the design in the MRPA leads to easy scalability for future soft- and hard- IP design.

5.1.2 Exploiting Data Statistics

For the HOG-based pedestrian detection co-processor, the truncation of the bit precision and non-normalization lead an average loss of 0.05% and 7.5% in true positive per window (TPPW) and true negative per window (TNPW) for INRIA dataset and an average loss of 1.5% and 8.2% for NICTA dataset with individual HOG features. The complementary dual-feature space is found to achieve an average of 95.3% in TPPW and 99.4% in TNPW for INRIA dataset. At the same time, NICTA dataset achieves an average accuracy of 96% in TPPW and 99.2% in TNPW.

5.1.3 Test Chips

A proof-of-concept prototype chip fabricated in 65 nm SOI CMOS, having thin gate oxide and BOX (Buried Oxide) layers (SOTB CMOS), with 1.96 mm^2 core area achieves an energy efficiency of 906 PJ/pixel and a processing speed of 30 fps for 1024 × 768-pixel image frames at 200 MHz recognition working frequency and 1 V supply voltage. Furthermore, multiple chips can implement image scaling since the designed chip has image-size flexibility due to the pixel-based architecture. Detection accuracy can be improved using complementary features in addition to the HOG feature, at the cost of an extra 40% power consumption, 64% area requirement, and 53% memory size.

The designed reconfigurable pipeline with parallel p-word input (R-PPPI) architecture for LVQ was taped-out using 180 nm CMOS technology with parallel 8-word inputs and 102 K-bit on-chip memory. The prototype achieves low power consumption of 66.38 mW (at 75 MHz and 1.8 V) in an area of 7.89 mm^2. In addition, I upgraded the 1st generation by a new modular and reconfigurable pipeline architecture (MRPA). The MRPA removes the dedicated learning circuits and expands the word-parallelism to 32 with 609 K-bit SRAM. Prototype fabrication in 65-nm CMOS technology achieves high-density efficiency and memory utilization efficiency with a core area of 2.14 mm^2, and average power consumption of 9.4 mW at 100 MHz and 0.8 V supply voltage.

5.2 Future Directions

I hope after several decades, the generation takes the computer vision for granted. With the futuristic mobile device, if they see some bug or a little crab on the beach, they will just assume that it has always been the case that they can just snap a photo and that system will tell them what it is and everything they could want to know about it.

5.2.1 Enhancing Pedestrian Detection Accuracy

Our group has tried to include complementary feature descriptor (Haar-like) for HOG to improve the accuracy. One future direction would be integrate more sliding-window based feature descriptors in one chip and vote on the basis of the detection results from all the descriptors. Besides, the current version has fixed primitive-size (cell, window, sliding step). So another direction is parametrize the primitive-size and the image size. In addition, enabling the learning function into the detection circuits can also improve the accuracy.

5.2.2 Embedding with Lane Detection

Lane detection has a great contribution on traffic-safety, as it is the major contributor to a lane departure warning (LDW) system, which is a basic and necessary part for an Advanced Driver Assistant System (ADAS). LDW is a system that uses the information from lane detection to warn the driver of lane departure. Then, the driver can correct the route to avoid that potential accidents happen. Due to its significance, many researchers pay attention and work on the study of LDW system for the advances in self-driving technologies. Accordingly, a real-time, robust and accurate lane-detection method is necessary for vehicle navigation. Combing the pedestrian detection and lane detection will result in a comprehensive result.

Appendix 1: TAOYAKA Onsite Team Project: Development of a lane detection system to improve the safety of visiting drivers

Project Title

Enhancing Tourism Development at Mitarai on Osakishimojima Island: Focusing on Inbound Tourism

Individual Report Title

Development of a lane detection system to improve the safety of visiting drivers

(Team 5)
Xiangyu ZHANG (D161378)

Overview of Research Area

Almost all the areas, which benefit from the rapid tourism growth, will experience an enormous increase in traffic accidents[1]. Before we go any further, we need to make as much preparation for the potential challenges as possible.

From several field trips, I found out that the Mitarai lacked in public transport. Only two bus routes[2] are available. One is from Okito Tenman-gu Shrine (沖友天満宮前) to Hiroshima Bus Center (広島バスセンター) (Fig. 1), and the other is from Okito Tenman-nan (沖友天満宮) to Chugoku Rosai Hospital (中国労災病院) (Fig. 2). Besides, the number of operating bus is very small. As a result, the promising tourists in the future will need to handle their transportation. The most likely transportation that will be chosen is a private car. With the boom

Fig. 1. The route map and timetable of the bus route from Okito Tenman-gu Shrine (沖友天満宮前) to Hiroshima Bus Center (広島バスセンター).

Fig. 2. The route map and timetable of the bus route from Okito Tenman-nan (沖友天満宮) to Chugoku Rosai Hospital (中国労災病院).

[1] Contemporary Perspectives on China Tourism by Honggen Xiao

[2] https://www.navitime.co.jp/poi?node=00164483

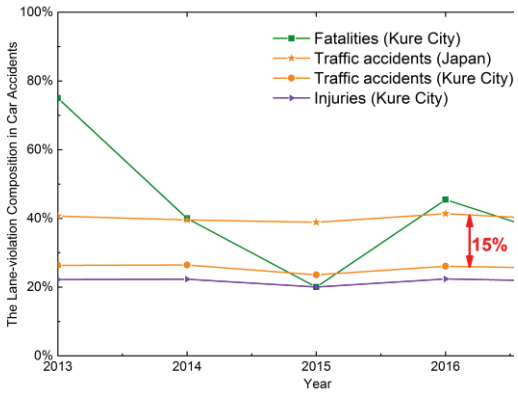


Fig. 3. The lane-violation accident is likely to increase 15% in Kure city with the tourism growth.

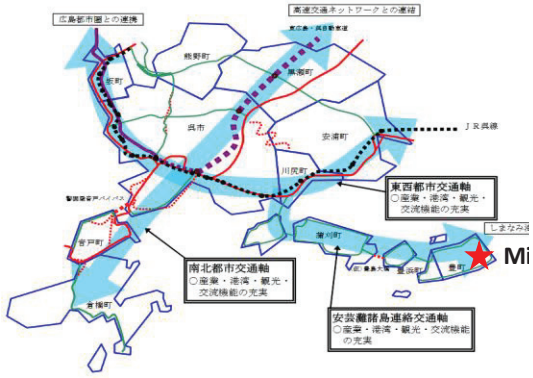


Fig. 4. The transportation map of Mitarai. The red star indicates the Mitarai. The blue lines represent the highway.

of tourism, the lack of public transport will add burden to traffic. We compare the car crash statistics of Kure city with that of Japan in Fig. 3[3]. We use the data of Kure city rather than Mitarai area because the overall traffic does influence the tourists who visit Mitarai by cars. The tourists need to drive through Kure city to reach Mitarai. Especially, there is only one main road from Akinadao Bridge to Mitarai (Fig. 4). We focus on the lane-violation because my research is about lane-detection that aims to avoid the violation. Fig. 3 shows that with the tourism development, the traffic safety will become more severe.

Another phenomenon (Fig. 5) that requires more consideration is the high ratios of senior citizen fatalities in car accidents. More than 50% deaths in car accidents occur in aged people. The figures are higher than in Europe and the United States.

In addition, foreigner tourists are more likely to cause traffic because of the differences of

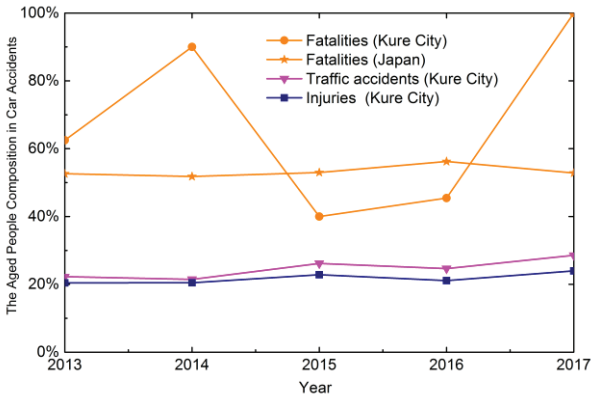


Fig. 5. Deaths among aged people exceeds 50% in car accidents.

[3] Hiroshima prefectural police: http://www.pref.hiroshima.lg.jp/site/police16/

driver's license systems, traffic rules, etc.[4]. Studies show that more than a half of the foreign visitors to Japan are “repeat visitors,” and one-quarter of them have been to Japan more than four times. These “repeat visitors” powerfully tend to drive cars by themselves and explore new destinations. Considering the local conditions of Mitarai, we expect that Mitarai will become a destination for the “repeat visitors.” The number of cars rented out to foreigners in Hokkaido and Okinawa of 2015 is 1.7 times higher than that of 2014. According to National Police Agency (NPA), there are approximately 800,000 foreigners who hold Japanese driver's licenses by 2015.

While much progress has occurred in the area of road safety, little attention has been paid to the safety of tourist drivers in Japan. The future of Mitarai’s tourism should be a path for sustainable growth. In order to ensure inbound tourism can grow sustainably, we need more robust infrastructure, capabilities, outreach, and internal collaboration. The traffic challenges not only affecting those going to work or school but also the development of industry and tourism.

Therefore, it is essential to devise effective and efficient measures to reduce traffic accidents causing by the tourism growth.

1.1.1 Traffic Safety along Tourist Routes

Previous research has investigated whether traffic safety is an issue for tourist drivers who visited the destinations with a number of tourism attractions and roadways to reach them. Wang et al.[5] concluded that when all other factors were equal, the tourism boom accounted for 15.8% more crashes on their research areas than that without tourism. Page et al.[6] found that in terms of rental car crashes, foreign drivers are the main sources of rental car crashes in New Zealand.

---

[4] Yoh K, Okamoto T, Inoi H, et al. Comparative study on foreign drivers' characteristics using traffic violation and accident statistics in Japan[J]. IATSS research, 2017, 41(2): 94-105. http://www.sciencedirect.com/science/article/pii/S0386111217300468

[5] Wang Y, Veneziano D, Russell S, et al. Traffic Safety Along Tourist Routes in Rural Areas[J]. Transportation Research Record: Journal of the Transportation Research Board, 2016 (2568): 55-63. http://www.montana.edu/ce/instructors_professors/faculty/Traffic_Safety.pdf

[6] Page S J, Meyer D. Tourist accidents: an exploratory analysis[J]. Annals of Tourism Research, 1996, 23(3): 666-690. http://www.sciencedirect.com/science/article/pii/0160738396000047

Petridou et al.[7] discovered that among the rental car crashes, the drivers from left-side driving country were more likely to encounter an accident. Petridou et al.[8] noticed that traffic crashes accounted for 40% of all accidents among foreign tourists on the island of Kerkyra in Greece. Leviäkangas[9] unearthed that crash rates of foreign drivers were much higher than that of local drivers.

1.1.2 Lane Detection

Lane detection has a great contribution on traffic-safety, as it is the major contributor to lane departure warning (LDW) system, which is a basic and necessary part for Advanced Driver Assistant System (ADAS)[10]. LDW is a system that uses the information from lane detection to warn the driver for lane departure. Then, the driver can correct the route to avoid potential accident happens. Due to its significance, many researchers pay attention and work on the study of LDW system with the advances in self -driving technologies. Accordingly, a real-time, robust and accurate lane-detection method is necessary for vehicle navigation.

The lane detection approaches can be mainly divided into two categories, sensors based approaches and image processing based approaches[11]. The sensors based approaches[12] use devices such as radar, laser sensors, and even global positioning systems (GPS) to mark the location of lane and vehicle by analyzing the captured information, and then estimate the lane departure occurs or not. These approaches are provided with high reliability in some bad

---

[7] Petridou E, Askitopoulou H, Vourvahakis D, et al. Epidemiology of road traffic accidents during pleasure travelling: the evidence from the island of Crete[J]. Accident Analysis & Prevention, 1997, 29(5): 687-693. http://www.sciencedirect.com/science/article/pii/S0001457597000389

[8] Petridou E, Dessypris N, Skalkidou A, et al. Are traffic injuries disproportionally more common among tourists in Greece? Struggling with incomplete data[J]. Accident Analysis & Prevention, 1999, 31(6): 611-615. http://www.sciencedirect.com/science/article/pii/S0001457599000172

[9] Leviäkangs P. Accident risk of foreign drivers—the case of Russian drivers in South-Eastern Finland[J]. Accident Analysis & Prevention, 1998, 30(2): 245-254. http://www.sciencedirect.com/science/article/pii/S0001457597000778

[10] Jung H, Min J, Kim J. An efficient lane detection algorithm for lane departure detection[C]//Intelligent Vehicles Symposium (IV), 2013 IEEE. IEEE, 2013: 976-981.

[11] Hoang T M, Hong H G, Vokhidov H, et al. Road lane detection by discriminating dashed and solid road lanes using a visible light camera sensor[J]. Sensors, 2016, 16(8): 1313.

[12] Yoo, Hunjae, Ukil Yang, and Kwanghoon Sohn. "Gradient-enhancing conversion for illumination-robust lane detection." IEEE Transactions on Intelligent Transportation Systems 14.3 (2013): 1083-1094.

weather conditions, but as their accuracy is not unreliable for detecting the lane positions, researchers usually combine sensors with some eigenvalue algorithms to increase the accuracy. The image processing based approaches use the features of captured images such as gradient[13], color[14], histogram[15], or texture (edge) for lane detection. Yoo et al.[16] propose a linear discriminant analysis (LDA)-based gradient-enhancing conversion method for lane detection aim at solving the problem that gradient values between lanes and roads vary with illumination change, which degrades the performance of lane detection systems. This method requires approximately 50 ms in each frame. Chiu et al.[17] propose another method based on color information. They used color-based segmentation with the quadratic function to find out and approach the lane boundary on a selected region of interest. As color segmentation is sensitive to ambient light and requires additional processing to avoid undesirable effects, it is not suitable for a road with the complex-light environment. Munajat et al.[18] combine RGB histogram filtration and boundary classification to describe a new approach for road detection. The RGB histogram filtration is used to process the input from camera by the color segmentation for determining the road area. Meanwhile, the boundary classification is used to map roads and its environments based on the RGB information. Finally, using Canny edge detection and Hough Transform (HT) to look for line boundaries.

The HT, one of the most robust image processing based approaches, has high reliability against line gaps or noise in real-world applications. Generally, to reduce computational cost of the HT based lane detection, the input image captured by camera is often down sampled by choosing a Region of Interest (ROI)[16]. The ROI contains the road lines in the original input image. Then, edge detection is implemented to prepare edge pixels for HT within the ROI. The number of edge pixels determines the computation amount of the HT. A suitable edge-detection algorithm

---

[13] Yoo H, Yang U, Sohn K. Gradient-enhancing conversion for illumination-robust lane detection[J]. IEEE Transactions on Intelligent Transportation Systems, 2013, 14(3): 1083-1094.

[14] Chiu, K.Y.; Lin, S.F., Lane detection using color-based segmentation, IEEE Proceedings. Intelligent Vehicles Symposium, 2005. IEEE, 2005: 706-711.

[15] Munajat M D E, Widyantoro D H, Munir R. Road detection system based on RGB histogram filterization and boundary classifier[C]//2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS). IEEE, 2015: 195-200.

[16] Li, X., Wu, Q., Kou, Y., Hou, L., & Yang, H. (2015, October). Lane detection based on spiking neural network and hough transform. In Image and Signal Processing (CISP), 2015 8th International Congress on (pp. 626-630). IEEE.

often lead to a better result of lane detection. Sobel[17], Canny[18] and Morphology[19] are the popular operator for edge detection. The function of HT is to confirm the polar coordinates corrected to a straight line.

The concept of the HT can be considered as a mathematical approach for gathering evidences in an accumulator array followed with a voting process for each events. Based on some mathematical rules, it defines a mapping process from Cartesian coordinate space to Polar coordinate (Hough space). Therefore, HT needs huge computing quantity and large memory usage. Due to these limitations, a software implementation for the HT based on general purpose CPUs is not suitable to real-time applications.

Focus on the operational process and the voting procedure of the HT, many improvements or optimization algorithms were proposed to improve the HT towards better suitability for real-time applications. Probabilistic Hough Transform (PHT) is one of the efficient improvements. PHT aims at increasing the operation speed of HT by the way of randomly selecting certain portions (choosing a subset) of the object points (edge pixels) to approximate the complete HT with a small as possible error for extraction of straight lines more quickly. It is observed that PHT can decrease the computational burden. However, although PHT can reduce resource consumption and processing time, it does not consider the location errors (errors between the actual line coordinates and digital image coordinates) so that the accuracy and the capability for anti-noise are worse than the original HT. Whereas, the lane detection has strict requirements on accuracy and noise.

1.1 Objective Hardware Architecture for Lane Detection based on Hough Transform (HT)

This study seeks to contribute to the safe travel for tourists who have the most possibility to visit Mitarai, the repeat visitors. Lane-violation is a major cause of car accidents. Considering that Hough Transform (HT) is an efficient way for lane detection as its high accuracy of

---

[17] Kortli Y, Marzougui M, Bouallegue B, et al. A novel illumination-invariant lane detection system[C]//Anti-Cyber Crimes (ICACC), 2017 2nd International Conference on. IEEE, 2017: 166-171.

[18] Rong W, Li Z, Zhang W, et al. An improved CANNY edge detection algorithm[C]//Mechatronics and Automation (ICMA), 2014 IEEE International Conference on. IEEE, 2014: 577-582.

[19] Li Q, Chen L, Li M, et al. A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios[J]. IEEE Transactions on Vehicular Technology, 2014, 63(2): 540-555.

straight-line detection. We develop a hardware architecture of HT with paralleled voting

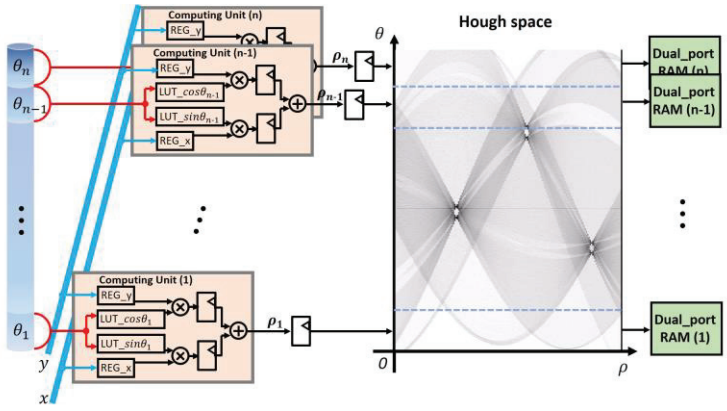


Fig. 6 Hardware implementation for Polar coordinates computation and voting procedure with n-fold parallelism.

procedure and local maximum algorithm for lane detection on an FPGA.

The lane detection system consists of four major modules: edge detection unit with Gaussian filter for removing noise, the computation unit of characteristic values (ρ,θ) for edge pixels, voting unit for each pair of (ρ,θ) with local maximum (ρ,θ) searching and the output of detected straight lines. The designed prototype system has been implemented on a DE1-SoC platform with a Cyclone V FPGA device. In the application of lane detection, the average processing speed of this HT implementation is 7.4 ms per VGA (640x480)-frame at 50 MHz working frequency.

1.1.1 Pipelined Computation and Parallelized Voting-Procedure

In the literature, the Coordinate Rotational Digital Computer (CORDIC) algorithm[20] is often used to calculate trigonometric functions for avoiding the multipliers. Given that the current FPGA device can provide sufficient multipliers, we use the look-up-table (LUT) solution for computation of sinθ and cosθ. The sinθ or cosθ fractional value, which are scaled by a certain factor in two's complement notation, are separately stored in n memory blocks. Meanwhile, n parallel units compute ρ as shown in the Fig. 6, where n is the divisor of the chosen number of discrete θ-values in Hough space. Each parallel part is composed of two multipliers, two LUTs (implemented as RAM) for sinθ and cosθ, one adder, and pipeline registers between the part-

[20] Andraka, R., A survey of CORDIC algorithms for FPGA based computers. 6th ACM/SIGDA International Symposium on FPGA 1998, 191-200.

internal computing units. In this paper, we adapt a factor of 8192 (2^{13}) and the computing unit is divided into 9 parallel parts.

1.1.2 Combination Method with Threshold Value Method and Local Maximum Searching

In the voting procedure, bins, each location (ρ,θ) in the Hough space are incremented for all lines that could pass through this location when an edge point is transformed to the polar space. The resolution of the Hough space determines the precision with which lines can be detected. The tradeoff between the memory usage of the Hough space and resolution parameters affects the system implementation for lane detection in hardware. Threshold value method (TVM) is the most popular way to find the polar coordinates in the Hough space for determining lines. All values above the threshold represent as lines. Due to the quantization error caused by resolution parameters Δρ and Δθ, the votes are usually distributed in a small area around the desired peak point as shown in Fig. 7. The static threshold cannot perfectly reproduce the original lines since many interfering straight lines around the real line.

The local maximum searching module is implemented by one dual-port RAM for storing the addresses of the centers, two counters, a state machine, and a few of logic elements as illustrated in Fig. 8. A global control signal "start" enables the state machine and the reading of the dual-port RAM. Each pair of (ρ,θ) ("sel_thetarho" in Fig. 8) and its vote value ("sel_outvalue" in Fig. 8) are the inputs to the Local maximum searching module. The local max estimation unit with three comparators triggers center updating according to the vote value (sel_outvalue). These states control the write enable of the dual-port RAM and the address

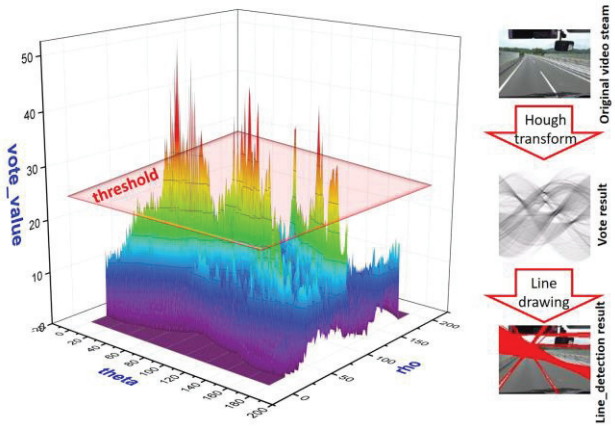


Fig. 7 Distribution of vote value in Hough space with $\Delta\theta = 1$ and $\Delta\rho = 1$.

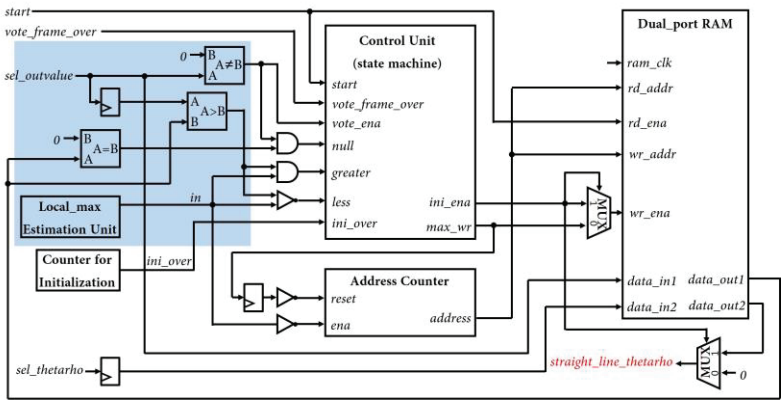


Fig. 8 Hardware architecture of local maximum searching algorithm for finding the potential

counter. Particularly, the input "sel_outvalue" is write in the dual-port RAM when a new center is updated. When the vote value of the center is larger than that of a new input (ρ,θ), the address counter is triggered for the comparison of a next address. The bit length of each word in the dual port RAM for storing the potential lines in Hough space is 28-bit. The MSB 10 bits (from 19th to 28th) are used to store the vote value of the centers; the next 8 bits (from 11th to 18th) are used for the corresponding θ and the LSB 10 bits are used to store the corresponding ρ. As we use paralleled structure to implement voting procedure, the Hough space is divided into n parts based on the range of θ. Therefore, the maximum number of the detected straight lines is n×W where W is the word number of each dual-port RAM.

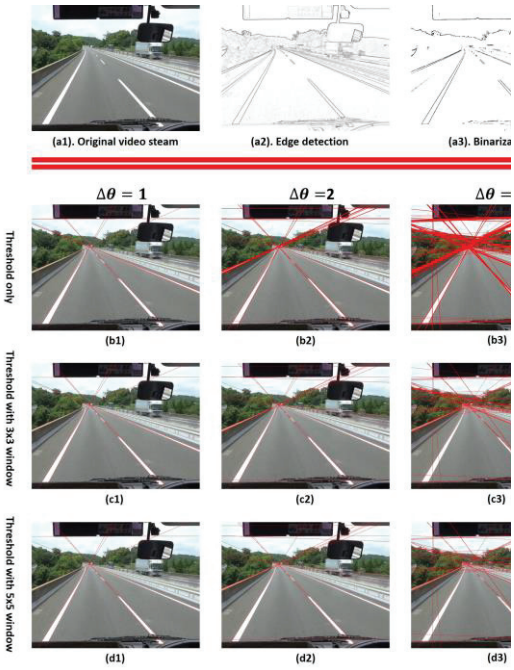


Fig. 10 Analysis of local maximum algorithm with different

1.2 Implementation Results

1.2.1 FPGA-based Prototype System for Lane Detection

A prototype system for lane detection implemented on a DE1-SoC board includes a D8M-GPIO camera module with 640×480 resolution at 60 fps, a Cyclone V (5CSEMA5F31C6) Altera FPGA device, and a LCD display as shown in Fig. 9. Besides the raw camera data capturing module, a pre-processing unit with a 3×3 Gaussian filter is designed for removing noise.

Before the implementation of lane detection on the FPGA board, we firstly have to define the discretization parameters (Δρ and Δθ) and the window size for local maximum searching. The resolution of the discretization parameters directly affects the memory usage of the Hough space. Furthermore, the incremental quantity for Δθ also determines the iteration number for each edge pixel. As shown in Fig. 10, after the

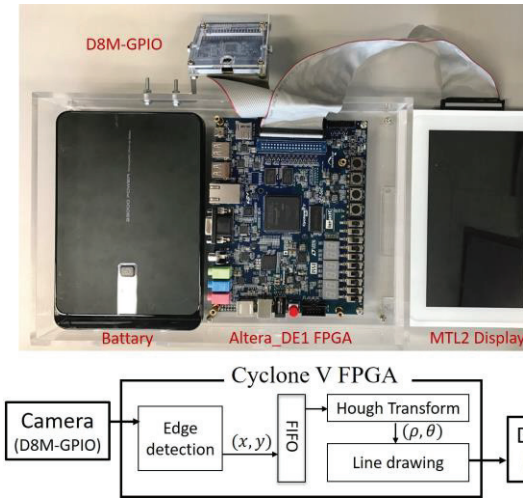


Fig. 9 Prototype system for HT-based lane

Table 1 Hardware resource usage for every module in the prototype system for line detection.

	Video capture	Pre-processing	HT	Display
Combinational ALUTs	2351	2113	3798	771
Registers	1371	2669	1476	192
Memory (Kb)	35	102	1775	26
DSP block	2	12	18	18
Max freq.			54.39	

edge detection (a2) by Morphology operator and binarization (a3),
different Δθ shows different the results of the straight-line detection where the threshold is set as (ρ_max/2). The results from (b1) to (b3) in Fig. 10 are attained by the threshold value method for finding the peak points as the outputs of the lines. The larger Δθ (from c1 to d1) causes many interfering lines so that the detection results become worse. Whereas, the local maximum algorithm can decrease the number of interfering lines effectively specially for the a large Δθ as shown in Fig. 10 (c2, c3, d2, and d3). In particular, the result with Δθ=2 and 5×5 sub-window for local maximum searching is better than that with Δθ=1 and thresholding method. Meanwhile, the memory usage in the case of Δθ=2 is only half comparing the case of Δθ=1. In general, the bigger size of sub-window for local maximum algorithm corresponds to the larger Δθ. However, this is also a tradeoff since the detection resolution for potential lines will become worse. Eventually, the incremental quantity Δθ is defined as 2 degrees and the sub-window size is set as 5×5.

The hardware resource usages of each function modules on DE1-SoC board with Altera Cyclone V FPGA device are listed in Table 1. The embedded system totally consumes 9033

Table 2 Comparison to the state-of-the-art works

	[1]	[2]	This work
Hardware platform	DE1-Soc	DE2	DE1-Soc
FPGA specification	Altera Cyclone V	Altera Cyclone II	Altera Cyclone V
working frequency	50 MHz	50 MHz	50 MHz
Image resolution	640×480	720×480	640×480
Processing speed (ms/frame)	41	40	7.4
Logic utilization	4694	-	5460
Combinational ALUTs usage	8600	14945	9033
Total memory bits (Kbit)	-	1555	1985
Embedded 18x18 multipliers	-	15	18
Total PLLs	-	2	2

combinational adaptive look-up tables (ALUT) that is a logical construct of the Cyclone FPGA device. The HT module occupies 42% logic utilization and 91.6% of the total memory usage with 1938Kb of the whole design. Finally, the maximum synthesized frequency is 54.39 MHz while an average processing speed of developed prototype system for HT-based lane detection is 7.4 ms per frame at 50 MHz.

We tested the system in a car around the HU campus.

1) The system can realize near real-time detection by offline processing.
2) The system is portable and can be fixed to a vehicle easily.
3) The lane detection result is of high accuracy, ensuring accurate reaction for the LDWS.

1.2.2 Analysis and Discussion

The speed performance and detect accuracy shown in Table 2 is compared to the state-of-the-art works[21], which are implemented on FPGA platform. A modified Canny-Hough lane detection algorithm is proposed in [1] for achieving real-time implementation. In order to reduce the complexity of algorithm, they simplified the gray conversion and Canny edge detection, separately. For the gray conversion, they used one-third of the sum of RGB to instead

[21] [1] Hwang, S., & Lee, Y. (2016, October). FPGA-based real-time lane detection for advanced driver assistance systems. In Circuits and Systems (APCCAS), 2016 IEEE Asia Pacific Conference on (pp. 218-219). IEEE.

[2] El Hajjouji I, El Mourabit A, Asrih Z, et al. FPGA based real-time lane detection and tracking implementation[C]//Electrical and Information Technologies (ICEIT), 2016 International Conference on. IEEE, 2016: 186-190.

of the multiple floating point operation. For the edge detection, they introduced three stages (Gaussian smoothing, Sobel edge tracking and sharpening) to simplify Canny edge detection. A super-resolution reconstruction algorithm was invited for selecting a ROI to decrease the time complexity of HT. As a result, the FPGA-based system achieves the processing speed of 41 ms/frame. Especially, they used an external SDRAM to store the converted gray-scaled image and the corresponding edge detection result so that this design requires a mountain of memory usage.

The robust lane detection and tracking system was presented in [2] which contains two main parts. One part is the Sobel operator with HT for lane detection through an adaptation, another part is the Kalman filter for dealing with lane tracking. They used Coordinate Rotation Digital Computer (CORDIC) algorithm to obtain $\sin\theta$ and $\cos\theta$ for the HT implementation. CORDIC is a simple and efficient algorithm to calculate trigonometric functions, which can avoid the memory of the sine and cosine initialization comparing with LUT method. However, the accuracy losses of CORDIC depends on the number of fractional bits and the number of iterations. Since the discretization parameters $\Delta\rho$ and $\Delta\theta$ are more coarse than that of this work due to the CORDIC computing, they used less memory than that of this work.

1.3 Conclusion

The inspiration from Mitarai and the onsite-team-project drove me to develop an embedded system for straight-line detection by Hough transform. The system is the major contributor to alane departure warning (LDW) system, which is a basic and necessary part for an Advanced Driver Assistant System (ADAS). LDW is a system that uses the information from lane detection to warn the driver of lane departure. With LDW, the driver can correct the route to avoid potential accidents. A combinational method of the thresholding and the local-maximum algorithm was presented for finding accurate lines with low resolution of the discretization parameters $\Delta\rho$ and $\Delta\theta$, leading to high computing speed and low memory requirement. A prototype system was designed on a low-cost Altera Cyclone V FPGA device to demonstrate the speed performance and the hardware efficiency.

Acknowledgment

Firstly, I would like to express my sincere gratitude to Prof. Hans Jürgen Mattausch for his continuous support of my doctor study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better mentor for my doctor study. Also, I would like to thank Prof. Idaku Ishii for his insightful comments, remarks and engagement through the learning process of this doctor research. Besides, I would like to thank Prof. Mitiko Miura-Mattausch, Prof. Shinji Kaneko, and Prof. Takeshi Takaki for their encouragement and guidance.

My sincere thanks also goes to Dr. Fengwei An, Dr. Lining Zhang, and Dr. Chenyue Ma, who gave access to the laboratory and research facilities. Without their precious support it would not have been possible to conduct this research.

I thank my fellow labmates in for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last 5 years.

Last but not the least, I would like to thank my family: my parents and my grandparents for supporting me spiritually throughout writing this thesis and my life in general.