UNIVERSITY OF CALIFORNIA
RIVERSIDE

Optimizing FPGA Design For Real Time Video Content Analysis

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

by

Xiaoyin Ma

March 2016

Dissertation Committee:

Dr. Walid A. Najjar , Chairperson
Dr. Amit K. Roy Chowdhury
Dr. Philip Brisk

The Dissertation of Xiaoyin Ma is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

It gives me great pleasure in acknowledging the support and help of my advisor Prof. Walid Najjar. This thesis should not have been possible without Prof. Najjar's guidance. He opened the window of using fixed-point arithmetic for computer vision applications on FPGAs, provided important suggestions along the way of my research and helped me developing my writing skills for publications. I have enjoyed every moment working with him.

Moreover, I would like to thank my committee members Prof. Amit Roy Chowdhury and Prof. Philip Brisk for giving me input on the latest development in various research fields and bringing fresh ideas to my research and publications.

To my wife, parents and family for all their support. I owe my deepest gratitude to my wife Wenjie Zhang whose encouragement led me through all the challenges and obstacles in the past few years.

ABSTRACT OF THE DISSERTATION

Optimizing FPGA Design For Real Time Video Content Analysis

by

Xiaoyin Ma

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, March 2016
Dr. Walid A. Najjar , Chairperson

The rapid growth of camera and storage capabilities, over the past decade, has resulted in an exponential growth in the size of video repositories, such as YouTube. In 2015, 400 hours of videos are uploaded to YouTube every minute [6]. At the same time, massive amount of images/videos are generated from monitoring cameras for elderly, sick assistance, satellites for earth science research, and telescopes for space exploration. Human annotation and manual manipulation of such videos are infeasible. Computer vision technology plays an essential role in automating the indexing, sorting, tagging, searching and analyzing huge amount of video data. Object detection and activity recognition in general are some of the most challenging topics in computer vision today. While the detection/recognition accuracy has increased dramatically over the past few years, it has not kept up with the complexity of detection/recognition tasks nor with the increased resolution of the video/image sources. As a result, the computation speed, and power consumption, of computer vision applications have become a major impediment to their wider use. Thus applications relying on real-time monitoring/feedback are not possible under current speeds. This thesis focuses on the use of Field Programmable Gate Arrays (FPGAs) to accelerate computer vision applications for embedded/real time applications while maintaining similar detection/recognition accuracy as the original processing. FPGAs are electronic devices on which an arbitrary digital circuit can be (re) configured under software control. To leverage the computational parallelism on FPGAs, fixed-point arithmetic is used for all implementations. The benefit of using fixed-point representation over floating point is the reduced bit-width, but the range and sometimes the precision are limited. Comprehensive studies are performed in this study to show that the classification system has some degree of tolerance to the reduced precision data representation. Hence FPGA programs are implemented accordingly in low bit-width fixed-point to achieve high computation throughput, low power consumption, and accurate classification.

As a first step, the impact of reduced precision is studied for Viola-Jones face detection algorithm: whereas the reference OpenCV [5] code uses double precision floating-point values, by using only five decimal digit (17 bits) fixed-point representation, the detection can achieve the same rates of false positives and false negatives as the reference OpenCV code. By reducing the necessary precision by a factor of 3X to 4X, the size of the circuit on FPGA is reduced by a factor of 12X; hence increasing the number of feature classifiers that can be fit on a single FPGA. A hybrid CPU-FPGA processing pipeline is proposed to reduce CPU work-load. As a second step, Histogram of Oriented Gradients (HOG), one of the most popular object detection algorithms, is evaluated by using the *full-image evaluation methodology* to explore the FPGA implementation of HOG using reduced bit-width. This approach lessens the required area resources on the FPGA and increases the clock frequency and hence the throughput per device through increased parallelism. Detection accuracy of the fixed-point HOG is evaluated by applying state-of-the-art computer vision pedestrian detection evaluation metrics. The reduced precision detection performs as well as the original floating-point code from OpenCV. This work then shows the single FPGA implementation achieves a 68.7x higher throughput than a high-end CPU, 5.1x higher than a high-end GPU, and 7.8x higher than the same implementation using floating-point on the same FPGA. A power consumption comparison for different platforms shows our fixed-point FPGA implementation uses 130x less power than CPU, and 31x less energy than GPU to process one image. In addition to object detection algorithms, this thesis also investigates the acceleration of action recognition, specifically a human action recognition (HAR) algorithm. In HAR, pedestrian detection is normally used as a pre-processing step to locate human in stream video. In this work, the possibility to perform feature extraction under reduced precision fixed-point arithmetic is evaluated to ease hardware resource requirements. The Histogram of Oriented Gradient in 3D (HOG3D) feature extraction is then compared with state-of-the-art Convolutional Neural Networks (CNNs) methods and result shows that the later is 75X slower than the former. The experiment shows that by re-training the classifier with reduced data precision, the classification performs as well as the original double-precision floating-point. Based on this result, an FPGA-based HAR feature extraction is implemented for near camera processing using fixed-point data representation and arithmetic. This implementation, using a single Xilinx Virtex 6 FPGA, achieves about 70x speedup over multicore CPU. Furthermore, a GPU implementation of HAR is introduced with 80x speedup over CPU (on an Nvidia Tesla K20).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid growth of camera and storage capabilities, over the past decade, and the related drop in their prices, has resulted in an exponential growth in the size of video repositories, such as YouTube. In 2012, 72 hours of videos were uploaded to YouTube every minute [6]. At the same time, massive amount of images/videos are generated from monitoring cameras for care to the elderly, assistance to the sick, satellite-based monitoring for earth science research, telescopes for space exploration, and security. Human annotation and manual manipulation of such videos is infeasible. Computer vision technology plays an essential role in automating the indexing, sorting, tagging, searching and analyzing huge amount of video data. Object detection and activity recognition in general are some of the most challenging topics in computer vision today. Despite significant progress in accuracy and speed over the past few years, the execution speed, less than one frame/sec in most of the existing methods, still limits real-time applications [98, 63] in a wide range of applications: health care, assisted living, surveillance, security, gaming, automobile, etc.

Moreover, many, if not most, monitoring situations increasingly rely on a network of cameras. These networks are mostly wireless for cost and portability reasons. Hence, the bandwidth is at a premium. It is therefore highly desirable to have the compute-intensive feature extraction stage done near the camera and to extract and transfer only action/object features and hence reduce network bandwidth requirements. The hardware acceleration of computer vision algorithms for those applications is particularly important due to the stringent power and speed requirement.

In the last few years we have witnessed the end of Denard Scaling that had held since 1974: It is no longer possible to increase the clock speed of digital devices simply by shrinking the feature size. While Moore's Law still holds, meaning that more cores can be built on the same die, a multi- or many-core execution suffers from high memory off-loading overhead for streaming data. Streaming video data needs to be loaded into memory before it can be processed. This implies that the processing speed will be limited by the memory bandwidth. FPGAs do not rely on memory offloading; rather the video data can be streamed directly onto the chip where it is processed. Furthermore, in recent years we have witnessed a tremendous increase in the size, speed and bandwidth capabilities of modern FPGA devices making them excellent candidates to implement, in hardware, large complex but massively parallel and bandwidth intensive applications such as action recognition.

Like many applications relying on numeric computations, computer vision applications make extensive use of floating-point number representation, both single and double precision. The major advantage of floating-point representation is the very large range of values that can be represented with a limited number of bits. Most CPU, and all GPU designs have been extensively optimized for short-latency high-throughput processing of floating-point operations. On an FPGA, the bit-width of operands in an application is a major determinant of its resource utilization, the achievable clock frequency and hence its throughput. By using a fixed-point representation with fewer bits, an application developer could implement more processing units on a given FPGA and each unit could achieve a higher-clock frequency because of its smaller footprint. However, smaller bit-width may lead to inaccurate or incorrect results. This research considers some of the most important computer vision algorithms for of their fixed-point analysis and implementation. As a first step, the three object detection (object detection is used as pre-processing step for action recognition) algorithms are examined for detection under reduced data precision including Viola-Jones face detection [93], HOG pedestrian detection [17] and aggregate channel features pedestrian detection [22]. Based on the fixed-point detection evaluation, FPGA implementations of Viola-Jones and HOG are also proposed with significant speedup over CPU, GPU and previous FPGA study. As a second step, the evaluation is expanded to video based feature extraction: HAR. The study of HOG3D algorithm is not only performed for reduced-precision feature extraction, but also applied to machine learning system to train the classifier using reduced precision features. Hence, complete end-to-end FPGA implementation for HOG3D HAR feature extraction is proposed in this work.

Face detection is one of the most widely studied computer vision algorithms that has a wide applications in commerce, privacy, security etc. With the emergence of high-resolution cameras, the computation complexity for face detection is increasing rapidly. Many algorithms have been proposed over the past decade in response to the increasing demand of fast face detection applications. Rowley [80] proposed a neural network-based face detection system. A color image face detection system was designed by Hsu [36]. Viola and Jones [93] introduced a cascade classifier face detection based on Haar-like features. In this algorithm, a detection window sweeps around the image to match the features of a face. It is one of the most popular methods in face detection that provides competitive detection rates in real-time detection applications. Viola-Jones algorithm is one of the most widely used computer vision algorithms. However, even with the optimized codes provided in Open Computer Vision Library (OpenCV) [5] the detection speed is far from satisfactory for real-time face detection applications. On the other hand, hardware based approaches such as FPGAs have been shown to be suitable to accelerate face detection. By using a dedicated hardware, face detection process can be parallelized to achieve maximum throughput. For example, Huang and Vahid [37] uses 16 classifiers executing in parallel to realize a detection rate of 110 FPS for 320×240 images using fixed-point representations of parameters to avoid expensive floating-point operations. Fixed-point representation constrains all parameters to a specific length and may lead to a loss of precision that might in turn affect the accuracy of the face detection. On an FPGA the size of the the circuit grows linearly with the size of the variables used. By reducing the number of bits used to represent values, the amount of resources used, such as LUTs and flip-flops, is reduced hence giving room to increase the parallelism of the circuit and hence its throughput. This thesis explores the effects of reduced precision on the accuracy of the face detection. Experiment results in this thesis show that the precision can be reduced from about 21 decimal digits to five decimal digits while keeping the same rates of false positives and false negatives. Programming FPGAs using low-level hardware description languages can be extremely tedious and time-consuming. ROCCC 2.0 toolset is used to generate the FPGA code of the Viola-Jones algorithm. ROCCC 2.0 is a C to VHDL compilation tool that relies on a subset of C to generate the hardware code and supports an extensive set of compile-time parallelizing optimizations.

Pedestrian/human detection is another important category in object detection and a very active research area. Pedestrian detection is commonly used for automobile or security monitoring applications. In these applications a high throughput and an economy of resources are highly desirable features allowing the applications to be embedded in mobile or field-deployable equipment. The HOG algorithm [17], developed for human detection is one of the most successful and popular algorithms in its class. In this algorithm, object descriptors are extracted from detection window with grids of overlapping blocks. Each block is divided into cells in which histograms of intensity gradients are collected as HOG features. Vectors of histograms are normalized and passed to a Support Vector Machine (SVM) classifier [91, 81] to recognize a person. The HOG algorithm was then expanded and incorporated into a multi-feature frame work known as aggregate channel features (ACF) [22]. ACF method combine several different features to form channels for significantly improved detection accuracy. The processing speed of ACF detection is faster than HOG as it uses features at one scale to estimate the features at other scales. In this thesis, the effects of reduced bit-width on the accuracy and performance of the HOG object detection algorithm is first evaluated by applying the full-image evaluation methodology and state-of-the-art computer vision pedestrian detection metrics. Using four sets of benchmarks, totaling 10,000 frames, it is shown that reducing the bit-width to 13-bits preserves the same detection accuracy as the original floating-point. The same evaluation metrics are also applied to the ACF algorithm by using three of the benchmarks to test the consistency of the "reduced-precision trick". Result shows that the ACF algorithm follows the same trend as HOG when the feature extraction precision is decreased. Then FPGA implementations of the HOG algorithm are proposed to explore the impact of reduced data precision on the area and clock frequency of the design. The throughput of the 13-bit fixed-point design (HOG-

Engine) on a single FPGA is then compared to that on CPU using floating-point (68.7x), a CPU with the Intel IPP library (60x), a high-end GPU (5.1x) and the same FPGA design using floating-point data (7.8x). The HOG-Engine uses a two-stage processing architecture to compute HOG feature extraction and SVM classification separately. Furthermore, the power consumption comparison for different platforms are discussed in this thesis.

For HAR, HOG3D algorithm [47] is chosen as the feature extraction method for its high accuracy and relatively low computational complexity. Compared to the state-of-the-art Convolutional Neural Networks (CNNs) methods, HOG3D is about 75x faster (See Chapter 5.4.4) . Therefore, the HOG3D method is more applicable for the real-time applications discussed above. In this work, in addition to the evaluation on the effect of reduced precision in feature extraction, the impact on machine learning/classifier training process is also investigated by training the classifier with data in reduced precision. Result shows that this training process can "compensate" the precision loss in feature extraction so that low bit-width can be used in the FPAG implementation. Furthermore, an end-to-end HOG3D feature extraction implementation is proposed. To the best of our knowledge, this is the first FPGA implementation that is targeted on full human body action recognition with state-of-the-art recognition rate.

The contributions of this thesis are:

1. A systematical test of the effect of data representations (fixed-point and floating-point) for Viola-Jones algorithmin the both false positive and false negative rates. The results show no difference in terms of rates after rounding the parameters to 5 digits. This result can lead to future fixed-point implementations of Viola-Jones face detection algorithm.

2. An FPGA development of Viola-Jones algorithm using ROCCC 2.0 toolset that dramatically saves time in timing and area constraint consideration. The ROCCC 2.0 based FPGA program is developed to parallelize classifier execution and speedup face detection. A frame work of hybrid executing Viola-Jones algorithm in both CPU and FPGA is proposed in this thesis. This method relies on an dual-FPGA-based parallel execution for the first 14 stages of Viola-Jones algorithm to reject, on average, 99.935% of all detection windows. The remainder of the computation is carried out by CPU. This method is capable of processing 3.1×10^{10} features per second which is equivalent to 403 frames of 640×480 images. It is 25 times faster than previous FPGA implementations [15]. At the same time, this implementation has the same accuracy compared to OpenCV implementations.

3. A complete experimental evaluation on the pedestrian detection accuracy of both HOG and ACF algorithm with fixed-point data using full-image evaluation as opposed to traditional per-window evaluation while varying the bit-width, using 10,000 benchmark frames with known ground truth.

4. A fully pipelined, two-step FPGA implementation of HOG algorithm is developed on a Xilinx Virtex-6 LX760 FPGA attached to Convey HC-2ex computer.

5. A comparison of the HOG3D throughput on FPGA, fixed and floating-point, CPU, with and without Intel IPP library, and the Nvidia Tesla K20 GPU, using 640×480 images at 1.05 scale factor, with bilinear interpolation and a window stride of four pixels (a low scale factor or window stride increases the detection accuracy but also the computational load).

6. A power consumption comparison between CPU, GPU and FPGA

7. A comprehensive evaluation of the HAR recognition accuracy under reduced data precision. The experimental results show that retraining the classifier using reduced data width can compensate for the precision loss in feature extraction and achieve the same recognition rate as using floating-point data. This result significantly relieves hardware resource requirement for video classification and enables faster and more energy efficient vision processing. The final FPGA implementation starts with 8-bit pixels but preserves the precision of the data using variable bit-width in the intermediate and final results.

8. A full FPGA implementation of feature extraction for real-time human action recognition targeting using near-camera processing. This implementation reads raw video pixels as input and produces the final bag-of-words features. The output bag-of-words is only 1,000 16-bit integers. Thus it is especially suitable for embedded platforms that process videos/images close to cameras to reduce network bandwidth requirement.

9. A throughput comparison of multicore CPUs, GPU and FPGA platforms shows the FPGA implementations in this work achieves 70x speed-up over multicore CPU while the GPU implementation achieves 80x speed-up.

The remainder of this thesis is organized as follows: Chapter 2 covers related work in the FPGA acceleration of object detection and HAR algorithm as well as the trade-offs between fixed and floating-point representations on FPGAs. Chapter 3 offers a detailed description on the Viola-Jones face detection algorithm and results on the proposed FPGA implementation. Chapter 4 discusses the fixed-point evaluation on the two pedestrian detection algorithm (HOG and ACF) and the FPGA implementation of HOG. Chapter 5 includes the detailed information on the assessment and experiments in HOG3D HAR algorithm.

Chapter 2

Background

2.1 Viola-Jones Face Detection

Many hardware accelerated solutions have been proposed for Viola-Jones face detection algorithm over the past few years. Both FPGAs and graphical processing units (GPUs) have been used as hardware accelerators. Gao and Lu [30] have designed the Haar-classifier on FPGA with retrained classifier features. Cho *et al.* [15] implemented a complete face detection system based on the AdaBoost algorithm with up to 16 frames per second (FPS) for VGA (640×480) images at scale factor of 1.2. By using 16 classifiers in parallel, Chen and Vahid [37] realized 110 FPS detection speed for QVGA (320×240) images. Cheng and Bouganis implemented a dynamic workload balancing method for FPGA based object detection [14]. This system has higher performance over resource ratio than Cho's result. Recently Qin and Zhu proposed a hybrid processing method on FPGA to parallel processing the first three stages and sequential processing later stages [78].

GPUs, another popular category of hardware accelerators, have been used to accelerate face detection. While parallelism in FPGA is implemented at the classifier level, GPUs compute multiple detection windows or sacles concurrently to achieve high throughput [72]. Hefenbrock *et al.* [33] proposed a multi-GPU implementation of the Viola-Jones face detection algorithm and got similar FPS as Cho's work on VGA images. Ore *et al.* [72] constructed a face detection system on NVidia GPU for real time HD videos and realized an FPS of 35 for 1080p resolution of Youtube videos.

Both GPUs and FPGAs have advantages and disadvantages. GPUs have high power consumption but are easier to program [33] and are convenient in using floating-point operations. On the other hand, FPGAs provide extreme power efficient but requiring substantial programming efforts in timing and area constraints. In addition, floating-point operations are expensive when used to perform a large number of parallel computations. As a result, fixed-point operations (integers) are used instead of floating-point [30, 15, 37] on FPGA implementations. However, fixed-point representations may lose precision.

This research studies how many decimal digits for classifier parameters are needed for an accurate detection which can direct further FPGA and hardware development. In addition, ROCCC 2.0 toolset is used to help generating hardware description language from a subset of C programming language. Moreover, the hybrid CPU-FPGA implementation result shows that the FPGA accelerated execution of Viola-Jones can be as high as 403 FPS for 640×480 images. The detailed experiments are shown in Chapter 3.

2.2 Pedestrian Detection

With the advent of computer vision algorithms in the past few years, various object detection algorithms have been developed to localize objects in images or video sequences. A sliding-window detection system based on Support Vector Machine (SVM) classifier was introduced in [75]. Based on this idea, the Viola-Jones object detection [93, 94] was proposed using the AdaBoost algorithm to train a cascade of classifiers; it has been reported as the most efficient method for object detection due to its use of integral images. In 2005, Dalal and Triggs proposed the HOG algorithm for pedestrian detection with a giant detection accuracy boost [17]. In this algorithm, pixel density gradients are computed and binned into overlapping blocks as the descriptor of objects. HOG and its variants are used extensively in modern computer vision applications [21, 22]. However due to its computational complexity, its application in real time detection is limited by execution speed.

The execution speed, or throughput, of HOG implementations is very strongly affected by (1) the frame size, (2) the scale factor, (3) the window stride, (4) the number of histogram bins, (5) the interpolation method used (e.g. bilinear, trilinear, or none) and the size of the region of interest. Since there is no one standard set of parameters, an objective comparison of performance across various implementations is difficult. Note that the scale factor, the window stride and the interpolation method affect the accuracy of the detection as well as the throughput. In this stduy, 640×480 frames are used, along with a 1.05 scale factor, a window stride of four pixels, nine histogram bins with bi-linear interpolation and a region of interest that is the whole frame. Starting with the widely accepted classifier in OpenCV [5], a fixed-point implementation of HOG is constructed in software to determine the optimal bit-width that does not compromise the detection accuracy while reducing the resource requirements. This fixed-point detection results is compared with the original floating-point result, by using four pedestrian detection benchmarks totaling 10,000 frames with known ground truth. Finally a fully pipelined FPGA accelerator is implemented with throughput comparison with those on state-of-the-art GPU and CPU.

Many hardware accelerated solutions have been proposed for HOG pedestrian detection, mostly using GPUs, with a reported speed-up of up to 67x [77, 88, 11, 104, 87, 66]. Because of deeply pipelined architectures and lower power consumption, FPGA platforms often provide higher execution speed and better energy efficiency over GPU [12]. An FPGA-GPU hybrid system was proposed in [9] using FPGA to extract HOG features and GPU to perform classification; it achieved a throughput of 10,000 detection windows per second for FPGA execution. Note that whole images (frames) were not tested.

In [57] a HOG feature extractor circuit for pedestrian and vehicle detection, using fixed-point data, was described with an estimated throughput of 33 fps at a single scale for 640×480 images. The detection accuracy was not reported or compared to a reference implementation. In [43], a HOG detection system was implemented on an Altera Stratix II FPGA using window size of 16×32 and scale factor of 1.2 achieving an estimated 30 fps for 640×480 video. Experiments in this work have shown that a scale factor 1.2 has 3.25x less computation than the 1.05 scale factor used in this paper and 6x poorer detection accuracy, in terms of true positives. In [12] a person detection execution on CPU, GPU and FPGA was compared for power, speed and accuracy. The FPGA implementation focused only on 4 out of 37 scales for 640×480 images and achieves 30 fps. In none of the papers above was the reduced bit-width used for HOG detection. A pedestrian detection system processing 18 scales of 1920×1080 resolution images at 64 fps was reported in [31]. Its throughput was estimated via simulation.

In [69], a real-time person detection was implemented on FPGA with a 62.5 fps on images with size equivalent to 320×240 at a single scale. While the data range of fixed-point values was reported (8-bits for input pixel, 19-bits for gradient, 14-bits for each histogram, and 33-bits for normalized histogram), there was no exploration of the tradeoffs in detection accuracy with reduced bit-width. Moreover, their fixed-point implementation showed a decrease in detection accuracy.

Mizuno *et al.* [68] have reported on the fixed-point parameter optimization by comparing the per-window detection results with the ground truth for INRIA person dataset [17]. The difficulty of INRIA benchmark is much simpler than those used in this paper. A fixed-point version of HOG detection for a digital signal processor (DSP) PICTOR was discussed in [101]. The detection accuracy was compared to with MATLAB's double-precision code without, however, reporting the bit-width. These approaches focused on comparing the detection window level output difference between fixed-point and floating-point computations. Nevertheless, per-window based evaluation methodology can fail to represent full image performance. For example many detected false positive windows in window-based evaluation can be removed by merging nearby bounding boxes in post-processing as shown in [21].

Scale factor (the ratio to scale the image after each detection) and window stride are one of the most important parameters in sliding-window based detection. Performing detection on the original scale can only find objects that have exactly the same size as the detector, thus multiple scale detections are necessary. Furthermore, a small window stride allows the detector to cover more possible object/person locations. It has been shown in [22] that the best detection performance can be achieved with a scale factor smaller than 1.09 and a window stride of four pixels. Scale factor and window stride together not only control how densely the detection window is applied across the image but also determine the computation complexity. To the best of our knowledge, all previous FPGA implementations have used a sparse detection, performing detection at a single scale, a subset of scales, large scale factors, or wide window strides to reduce the computational load but doing so also significantly compromises the detection accuracy [22]. Further, none of the previous FPGA implementations adopt the bi-linear interpolation.

ACF algorithm is an extension of the popular HOG algorithm that incorporates multiple feature types [22]. In this algorithm, pixel values, pixel gradients and HOG form 10 channels of features (three channels for color, one channel for gradients, and six channels for HOG) to describe an object. The feature values in each channels are then integrated similar to integral image [93] for fast feature computation and lookup. Since multiple features are used to describe target object, the detection accuracy of this algorithm is significantly better than HOG. The processing speed of ACF is also increased from HOG by predict the channel feature values in one scale from adjacent scales [21]. In this work, the fixed-point evaluation of ACF algorithm is similar to the HOG evaluation with the exception that Daimler benchmark is not used (as this algorithm requires color image to process).

2.3 Action Recognition

Various HAR algorithms have been developed over the past few years [76]. Most can be broadly described in terms of the following framework. A temporal sliding window is applied on a video stream to find actions. For frames within a window, spatial-temporal features are extracted from 3D regions by either interest point detector or dense-sampling.

The extracted features are clustered using K-means algorithm to build visual vocabularies in training. These features are then binned into histograms based on their center to form high-level fixed-size bag-of-words (BOW) feature vector [86, 16, 95, 99, 54]. A classifier is trained using BOW features to detect the targeted action. In many recent methods, the relationships between the actions are also exploited in a structural support vector machine (SVM) [103] or graph-modeling framework [107]. Note that this work only considers the case that each video clip contains one action and there are known number (and known labels) of actions to be recognized.

Despite many different detectors being developed for interest point detection [52, 19, 71, 41, 90, 100], it has been shown that sampling the window at regular positions in space and time (dense-sampling) achieves the best performance in most benchmarks [98]. Also many algorithms have been proposed for the abstraction of pixel information to capture human movements, including patches of normalized derivatives in space and time [82], image gradients [19], optical flow [62, 19], Speeded-up Robust Features (SURF) extended to 3D (eSURFT) [100], combination of histograms of oriented gradients and histograms of oriented optical flow [51], and 3D extended HOG (HOG3D) [47, 46]. Among the various spatial-temporal features, HOG3D with dense-sampling has been shown to achieve good performance for HAR [98] while being less stringent in hardware requirement. Thus, this method in our real-time embedded action recognition system.

Previous FPGA implementations have mainly focused on the hand gesture recognition, a predecessor of HAR [84, 58]. The design of a vision processing chip that can be used for gesture recognition is described in [65]. In [35] a 600-fps real time action recognition system was proposed for four types of hand gestures. Meng and Freeman implemented a reconfigurable system for action recognition and have tested the algorithm using a full human body action benchmark [67]. However, the 63% mean-average recognition rate on the KTH dataset [82] is well below state-of-the-art results. To the best of our knowledge, none of the previous work have performed action recognition under reduced data precision. This thesis proposes an FPGA implementation of HAR that is comparable to state-of-the-art recognition rates in complex HAR benchmarks while maintaining efficient FPGA resource usage by applying 8-bit fixed-point arithmetic.

Previously, the study of fixed-point implementation in image and video processing were primarily focused on the data range analysis and precision/errors assotocated the reduced bit-width [73, 56, 10]. It was also shown that machine learning model has certain tolerance on the reduced data precision and has lead to various FPGA implementation using fixed-point data [101, 68, 63] in the object detection work. In HAR implementation, a learning predictive model is built using the reduced precision features and it is shown that the new model can compensate for the precision loss in feature extraction having a comparable recognition rate with double-precision floating-point.

2.4 FPGAs and Fixed-point Representation

FPGAs are integrated circuits that can be program by a customer. FPGA consists of hundreds of thousands of small (3 to 6-input) programmable Look Up Table (LUTs) that are used to implement boolean functions. Figure 2.1 shows a 2-input LUT configured as an AND gate. LUTs can be used to represent complicated logic when combined together.

As shown in Figure 2.2, logic function f(A,B,C) = (A AND B) OR C can be implemented using two 2-input LUTs. For the purpose of more generic platforms, this is achieved through the use of the configurable interconnects, also known as switch matrices.

Hardware designers construct circuit functionalities in a hardware description language, such as VHDL or Verilog. The HDL programs are passed through a complex tool chain that analyzes the circuit description, optimize it for the FPGA at hand, and map it to the available hardware resources (called synthesis).

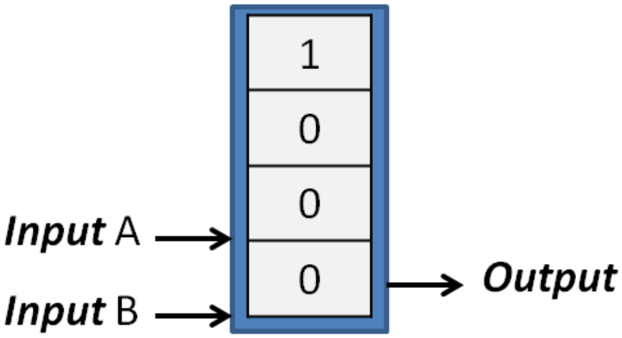

Figure 2.1: Implementing a 2-input AND gate using a 2-input LUT.

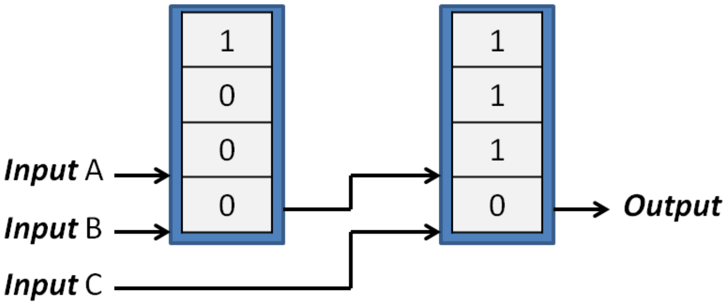

Figure 2.2: Implementing *f(A,B,C)=(A AND B) OR C*, a 3-input boolean function, using two 2-input LUTs.

The performance advantages of such platforms arise from their ability to execute thousands of computations in parallel, relieving the application at hand from the sequential limitations of software execution on Von-Neumann based platforms. Moreover, like software programs, FPGA programs can be modified and reconfigured to adapt changes in algorithms. Specifically in this case, the program can be changed for other object detections.

While FPGA platforms can process streaming data with hundreds of times speedup over traditionally CPU based processing, it is very expensive to perform arithmetic operations in the look-up table based architecture such as all floating-point operations and some fixed-point based operations.

Floating-point and integer are the most common data types in computer programs. The main advantage of a floating-point representation, given a fixed bit-width, is its very large range; its precision, however, deteriorates as the value represented grows. Fixed-point values are essentially integers with a fixed place of radix-point. Their range is determined by the number of bits to the left of the binary-point while the precision is determined by those to the right of it. Arithmetic operations on floating-point values require careful manipulation of the mantissa and exponent as well as rounding, normalization and re-normalization. All of these steps are hidden away from the programer by hardware floating-point units on all CPUs and GPUs.

On an FPGA, smaller bit-width is desired to achieve higher clock frequency and fewer resource usage. To show how different data types will affect FPGA resources, the classifier parameter length and their effect on FPGA resources are tested. The stump-based cascade classifier (used in the Viola-Jones implementation this thesis) were constructed by Lienhart *et al.* [59]. It has 2135 Haar features divided into 22 stages with a 20×20 detection window. Five FPGA programs are constructed to compute the first 4 stages of Viola-Jones face detection algorithm by using parameters in double precision floating-point, 20, 7, 6, and 5 decimal digits of fixed-point respectively. Double precision floating-point is used in the OpenCV implementation [5]. To fully represent the original precision in fixed-point, 20 decimal digits are used (64 binary bits) to represent the parameters. Also all values are rounded to 7 (24 bits), 6 (20 bits), and 5 (17 bits) digits for comparison in our FPGA programs. The results, Figure 2.3, show that double precision floating-point uses 12

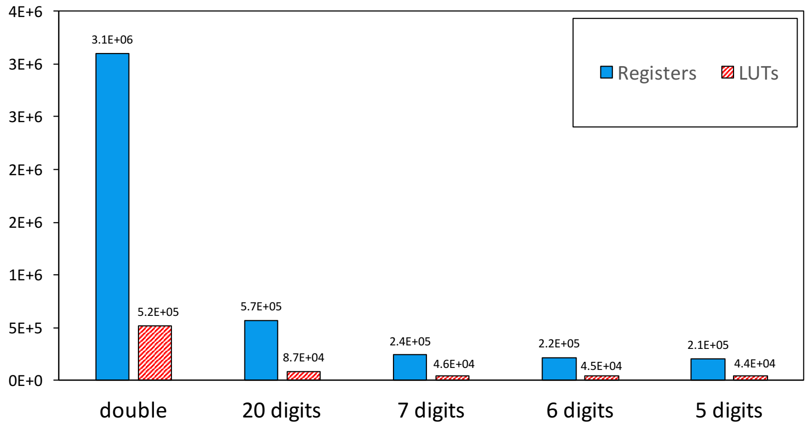

Figure 2.3: FPGA resource utilization comparison between different parameter sizes. Note that the 20 digits is the same precision in fixed-point as the double precision flowting-point.

times more LUTs and 15 times more registers on the FPGA the 5-digit representation. The 20-digit fixed-point representation uses twice as many LUTs as the 5-digit one.

To further evaluate the impact of bit-width and different operations on FPGA resource utilization, the FPGA resource usage of single unit addition and multiplication in different bit-width for both fixed-point and floating-point are evaluated as shown in Table 2.1. All values are obtained using Xilinx Virtex-6 LX760 FPGA with ISE 14.3 after place & route. Xilinx CoreGen generates all floating-point units and fixed-point multiplication units. 64-bit, 32-bit, and 16-bit floating-point are IEEE standard. 13-bit floating-point has five exponent bits and eight fraction bits. As shown in Table 2.1, fixed-point additions use 10.5-12.5x less LUTs than floating-point addition while operate at 1.6-2.4x higher frequency. Furthermore, floating-point additions require more registers as the computation takes several clock cycles. Fixed-point multiplication requires more FPGA area than floating-point multiplication since the product of two 32-bit integer multiplication is 64-bit while the result of two 32-bit floating-point multiplication will yield another 32-bit value, as shown in

Table 2.1: FPGA resource utilization between fixed-point and floating-point data.

Fixed-Point Results						Floating-point Results					
# Bits	Reg.	LUTs	DSPs	Latency	f (MHz)	Bits	Reg.	LUTs	DSPs	Latency	f (MHz)
Fixed-Point Addition [1]						Floating-Point Addition					
64	130	76	0	2	235	64	1034	800	0	12	268
32	66	36	0	2	541	32	541	397	0	12	390
16	34	20	0	2	627	16	224	171	0	8	397
13	18	13	0	2	609	13	193	142	0	8	412
Fixed-Point Multiplication without DSP						Floating-Point Multiplication without DSP					
64	4296	4293	0	6	219	64	2431	2309	0	9	179
32	1098	1099	0	5	345	32	681	634	0	8	226
16	279	283	0	4	438	16	202	185	0	6	353
13	216	194	0	4	445	13	151	129	0	6	396
Fixed-Point Multiplication with DSP						Floating-Point Multiplication with DSP					
64	859	437	16	18	308	64	391	308	10	15	291
32	53	2	4	6	473	32	179	132	3	8	325
16	4	1	1	3	473	16	89	74	2	6	398
13	0	0	1	3	473	13	80	64	1	6	370

Table 2.1. However, the multiplication of small bit-width values can take the advantage of on-chip DSP block to ease the area usage. Table 2.1 shows the FPGA resource utilization when using DSP block for both multiplications. 32-bit and below fixed-point multiplication benefit from the usage of DSP blocks.

Fixed-point arithmetic uses less FPGA area and runs at a higher frequency than floating-point operations. Hence, with sufficient memory bandwidth, one can place more fixed-point modules on a single FPGA running at higher frequency to increase the overall throughput. However, the use of fixed-point data may compromise the accuracy of the detection.

[1] The latency for a regular fixed-point adder should be one. An additional output stage is intentionally added here to obtain correct timing results.

Chapter 3

Acceleration on High Accuracy Face Detection

3.1 Viola-Jones Face Detection Algorithm

Viola-Jones algorithm was introduced in 2001 as a fast face detection algorithm. In the Viola-Jones algorithm, Haar-like features are used to construct face information [74]. Haar-like feature consists the weighted sum of two or three adjacent rectangular areas as shown in Figure 3.1. The equation to compute Haar-like features are shown in Equation 3.1 where (S_i) is the pixel sum of ith rectangular area and W_i is its weight.

$$H = S_1 * W_1 + S_2 * W_2 + S_3 * W_3 \tag{3.1}$$

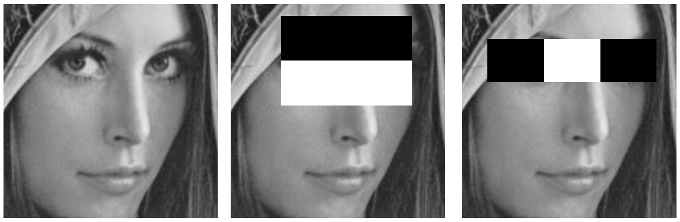

Figure 3.1: Illustration of Haar-like features in a detection window.

To fast compute the area sums, the concept of integral image is proposed in Viola-Jones algorithm. The integral image $(I(x', y')$ at pixel $P(x', y')$ is computed using Equation 3.2.

$$I(x', y') = \sum_{0}^{x'} \sum_{0}^{y'} P(x, y) \tag{3.2}$$

By using integral image, the pixel sum $S(x, y, w, h)$ at a rectangular region (x, y, w, h) (as the top left point coordinate value and the width, height of the rectangle) can be fast computed using Equation 3.3. Note that for any negative indices in Equation 3.3, the value of the integral image values are 0.

$$\begin{aligned} S(x, y, w, h) =& I(x + w, y + h) \quad && + I(x - 1, y - 1) - \\ & I(x - 1, y + h) \quad && - I(x + w, y - 1) \end{aligned} \tag{3.3}$$

Also note that to enable light (contrast) correction, all pixel values are normalized as in Equation 3.4.

$$\bar{I}(x, y) = \frac{I(x, y) - u}{\sigma} \tag{3.4}$$

The normalization can be performed either before the integral image or after the integral image (depending on implementation). In this work, the normalization is done at the time of computing area sum.

Haar features are extracted from a 20×20 detection window by a classifier to determine if the window is a face. The detection windows is moved across the image at a stride of 1 pixels to cover all possibly locations of a face. In addition, to detect faces with different sizes, the original image is re-scales $\sqrt{2}$ times each time until the image is smaller than the window-size.

A cascade classifier is designed in Viola-Jones algorithm to fast reject false detection windows. The classifier consists of 22 stages. Each stage is a combination of three to 200 simple classifiers. At the first stage (stage 0), only three features are used. As the stage number becomes larger, the classification becomes more stringent meaning that it is more difficult for a window to pass that stage. When a window passed all 22 stages of classifier, the window is considered as a face. As the classifier is in a cascaded structure, parallel processing multiple stages at the same time can accelerate the rejection of a detection window. Thus, the following sections will be focused on the parallel processing (and fixed-point operations) in the classifier.

3.2 Face Detection at Reduced Parameter Precision

This section describes the impact of a reduced precision in fixed-point on the accuracy of the face detection. Three face detection benchmarks are used to test the reduced-precision classifiers [39, 40, 1].

Information for the three benchmarks is in Table 3.1. Three benchmarks with different detection difficulty are used to evaluate the modified classifiers. There are total of 11,692 faces.Face detection data set and benchmark (FDDB) [39] uses a subset of images from the Labeled Faces in the Wild dataset [38] (LFW). Face images in the LFW data set have a large variation in clothing, background, and other variables. The evaluation of detection result was performed with the tool in the benchmark. BioID Face Detection Database (BioID) consists 1521 images of human faces with varying illumination and complex background. Face and Gesture Recognition Working group [3]

Table 3.1: Face detection benchmark image information.

Benchmark	# of Faces	Resolution	Faces per Image
FDDB[39]	5171	$\leq 450 \times 450$	1 or multiple
BioID[40]	1521	384×286	1
Talking[1]	5000	720×576	1

manually placed 20 feature points on each face. Talking Face Video (Talking) [1] data set has 5000 frames taken from a video of a person in conversation. Each face is annotated by 68 points. Both BioID and Talking data sets are evaluated by a program that if 60% of the points fall inside the detected face region, the detected object will be counted a face. Duplicate detection will only be counted once. The original classifier has 16-22 decimal digits parameters. These parameters are rounded to 7, 6, 5, 4, 3 decimal digits respectively to perform face detection with the three benchmarks. Results indicate that rounding to 5 decimal digits achieves the same accuracy as the original classifier. Detection accuracy results are shown in Table 3.2. After rounding the parameters to 5 decimal digits one more false negative was observed in BioID dataset but two fewer false positive were seen in the FDDB dataset. The performance of this classifier is almost the same as the original one. In addition, 3 and 4 digit precision show significant increase in false positive rate, especially for Talking dataset. As a result, in FPGA implementation, the five-digit fixed-point representation of classifier data can be used instead of original double precision floating-point while retaining the same detection accuracy.

Table 3.2: Face detection accuracy as function of precision for the three benchmarks.

	False Negative			False Positive		
# of Digits	FDDB	BioID	Talking	FDDB	BioID	Talking
3	1322	48	0	339	105	1605
4	1296	51	0	277	45	108
5	1297	52	1	263	42	41
6	1298	51	1	266	42	41
7	1297	51	1	264	42	41
original	1297	51	1	265	42	41

3.3 FPGA Implementation Using ROCCC 2.0

The traditional development of FPGA applications relies on extensive hardware design experience, careful implementations and attention to details to achieve the target timing. The goal of C to hardware tools is to reduce that burden and make this process accessible to traditionally trained applications developers. This implementation uses Riverside Optimizing Compiler for Configurable Computing [92] (ROCCC 2.0) tool for FPGA development. The ROCCC 2.0 toolset will be briefly discussed and then implementation details will be shown.

3.3.1 ROCCC 2.0 Toolset

ROCCC 2.0 is a C to HDL compilation tool focused on FPGA-based code acceleration. Its objectives are to maximize parallelism within the constraints of the target device. ROCCC 2.0 toolset uses a sub-set of C programming language and generates VHDL code for FPGA. ROCCC

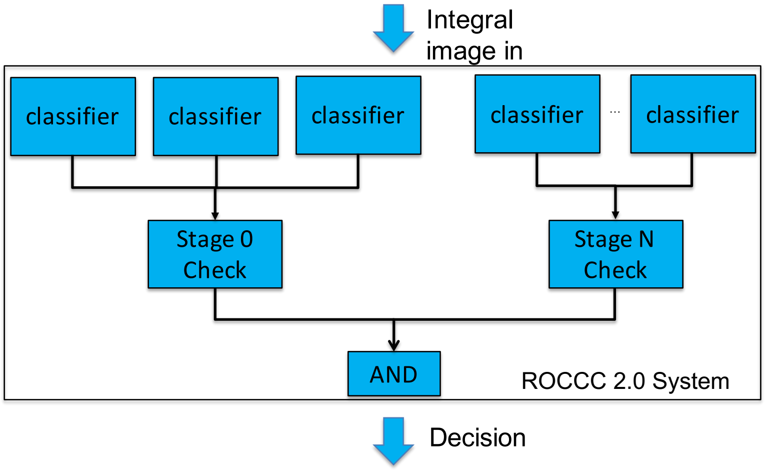

Figure 3.2: Viola-Jones face detection system generated by ROCCC. In ROCCC generated systems, all computations are executed in parallel unless dependencies exist. For example, all classifiers are running at the same time, and stage check will be executed after getting result from classifiers.

2.0 toolset dramatically saves FPGA development time and automatically parallels all classifier executions to achieve maximum performance.

The architecture of the face detection system generated by ROCCC 2.0 is shown in Figure 3.2. In this system, all classifiers are evaluated at the same time to achieve the highest possible throughput. This design is similar to Qin's work that uses mulitple classifiers for the first three stages (39 classifiers) [78]. However, unlike previous work, this thesis do not perform sequential processing on hardware. All remaining processing that hardware can not fit for parallel processing are performed in multi-threaded software evaluation. All computations on FPGA are pipelined so that evaluation of detection windows can be overlapped. Other configurations such as the number of scales, and minimum face size can be specified by the host program that directs the communication between FPGA and the host computer.

3.3.2 Parallel Execution in Hardware

The FPGA implementation design in this thesis includes classifier evaluation, and stage check. This work does not address the generation of the integral image on an FPGA. The software host code gets the images, from disk or a video stream, computes the integral image, scales it and sends it to the FPGA.

Integral image and standard deviation are computed by CPU, and stored in memory. Memory controller reads a detection window at a time and sends to classifier for evaluation. In order to avoid stall in the parallel evaluation, multiple detection windows are stored in memory before starting execution. In the parallel execution, evaluation of each detection window has fixed latency, as a result, the produced result can be matched with corresponding position and scale in an image by a computer program.

Previous FPGA implementations only perform integral image for the detection window. Those designs use 17 bits signal (each pixel value will not be larger than $2^8 \times 400$) for integral image can save some resources on hardware. However, it results multiple computations of the same pixel as the detection window sweeps around the image. In this design, the FPGA classifier takes a general 32-bit integral image value for evaluation. Thus, the integral image for the entire image can be generated at once without re-computing the same pixels multiple times. Experiment in this thesis indicates 32-bit classifier uses about 1.2 times more registers than the 17-bit one and uses almost same number of LUTs. But the computation of integral image is significantly reduced since each pixel is included in multiple detection windows.

In previous FPGA implementations, integral image is computed only for the detection window. As a result, each integral image value will never be larger than 2^{17} (each pixel $2^8 \times 400$).

Table 3.3: Area comparison for 17-bit and 32-bit classifier.

Classifier	LUTs	Registers
17 bits	43960	207303
32 bits	53448	255854

This saves certain hardware area for classifiers but results in duplicated computation of the integral image. In this implementation, classifiers take integral image values computed for an entire image to avoid redundant computations, but uses 1.2 times more hardware resources. The comparison of hardware area is shown in Table 3.3

3.3.3 Hardware Resource Constraints

The level of parallelism on FPGA is limited by the available resources on each device. To explore area utilization of proposed FPGA architecture, VHDL code were synthesized on a target FGPA chip Xilinx Virtex-6 LX240. The synthesis result indicates that a Xilinx Virtex-6 LX240 FPGA can fit the first 10 stages (384 32-bits classifiers in parallel) of Viola-Jones face detection algorithm. The FPGA program is pipelined and running at 81M Hz frequency. Thus it can execute 8.1×10^7 detection windows (3.1×10^{10} features) every second. This processing speed is equivalents to 403 640×480 images at a scaling factor of 1.2. At least six Virtex-6 LX240 FPGAs would be needed to implement all the stages of the Viola-Jones algorithms.

To optimize hardware resource usage, this work investigates how much parallelism are actually needed for Viola-Jones face detection. Each one of the 22 stages consists a group of weak classifiers that becomes stronger at later stages. A statistical test on the detection process is performed to see how many detection sub-windows are passed at each stage. The tests are based on the three previously used face detection benchmarks with 1.1 scale factor and minimum object size of 30×30 pixels. A total of 3,063,452,070 detection sub-windows are evaluated. The statistical result of passed windows is shown in Table 3.4. According to this result, 0.065% of all sub-windows can pass the first 14 stages (stage 0-13). What's more, the first 14 stages only use 36% of the total computation resources (780 out of 2135 features). Thus it is conceivable to have a two-FPGA system where the first 14 stages (rejecting 99.935%) are in parallel and the remaining stages computed iteratively on FPGA or using a multi-thread CPU program to process later stages.

To evaluate the speedup of this approach comparing to software execution, a single threaded Viola-Jones OpenCV program is tested on an Intel Xeon E5540 CPU with 36GB RAM. The software execution time is shown in Table 3.5. On average, CPU can process 4.06×10^7 features every second with an average FPS between 2 and 8. FPGA execution speed is about three magnitude faster than CPU. Therefore, the proposed architecture is a promising method to speedup face detection algorithm.

3.4 Results and Discussions

The FPGA implementation described in section 3.3 was on a Pico M501 machine with a Xilinx Virtex-6 LX240 FPGA. The architecture of the Pico FPGA is shown in Figure 3.3. The host computer is connected to the Pico board via x8 gen2 PCI Express bus for data communications. The FPGA program and hardware interface is generated by ROCCC 2.0. A C++ program on the host

Table 3.4: Cumulative percentage of sub-windows that pass each of the 22 stages.

Stage	Cumulative %	Stage	Cumulative %
0	75.151	11	0.164
1	40.054	12	0.102
2	21.529	13	0.065
3	11.329	14	0.043
4	9.107	15	0.033
5	5.124	16	0.027
6	2.486	17	0.022
7	1.432	18	0.020
8	0.736	19	0.018
9	0.433	20	0.017
10	0.284	21	0.016

Table 3.5: Software execution time for the three benchmarks.

Benchmark	Time (s)	FPS	features/sec
FDDB	526.86	5.4	3.95×10^7
BioID	202.99	7.5	4.20×10^7
Talking	2794.7	1.8	4.03×10^7

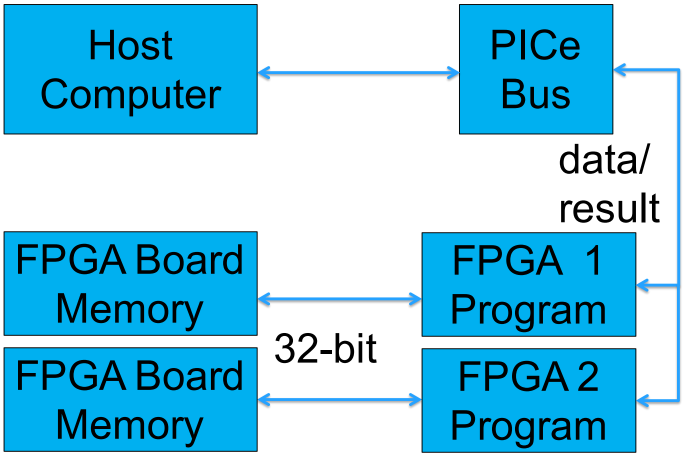

Figure 3.3: Diagram of Pico machine architecture. Host computer will perform integral image and sdandard deviation and send data to the FPGA DDR3 Ram via PICe bus. FPGA will execute the detection algorithm and send result back to CPU.

Table 3.6: Final implementation result on Xilinx Virtex-6 LX240 FPGA and speed.

Stages	LUTs%	Register%	Frequency (MHz)	FPS
0-9	47	42	97	482
10-13	49	44	81	403

computer calling Pico Application Programming Interfaces (API) controls the communication between FPGA and CPU. On the FPGA side, a 513MB DDR3 memoryis installed on board providing buffers for the integral image data.

The FPGA program implements the parallel version of feature evaluation. As shown in Table 3.6, the first 10 stages with 384 classifiers can be fitted on a single FPGA. By using a two-FPGA design, 780 classifiers can be executed in parallel in the first 14 stages. Two FPGAs running at 81MHz frequency can compute 8.1×10^7 detection windows per second. As the statistical data shows, only 0.065% of sub-windows will pass the first 14 stages. Therefore, later stages will need to have an execution speed of 9.2×10^7 features per second (under worst case if all passed windows will need to be processed for stage 14-21). This requirement is 2.3 times faster than a single thread CPU can compute. However, this limitation can be easiliy overcome by using multiple CPU cores program to evaluate the rest of the stages if necessary. As a result, the proposed face detection frame work is suitable for real time high accuracy detection.

3.5 Conclusion

In this chapter, the parameter precision on the effect of detection result in Viola-Jones algorithm is tested systematically. The results show that after rounding parameters to five decimal digits, no difference were observed in detection accuracy. In addition, a hybrid execution of Viola-Jones algorithm is proposed by using hybrid FPGA and CPU system to accelerate the computation. In this frame work, for each detection window, the first 14 stages (stage 0-13) are executed in parallel on FPGAs. If a window passes, it will then be processed in CPU by multi-thread program. Experiments have shown that only 0.065% of the detection windows in face detection will need to be processed in stage 14-21. As a result, the FPGA parallelization significantly increases detection speed. Finally the prototype FPGA program is evaluated on a Xilinx Virtex-6 LX240 FPGA.

The result on detection precision can be used for future FPGA development of Viola-Jones face detection algorithm. In addition, the parallel implementation can be used for other image processing techniques that require high computation complexity. Future work involves refines the integral image computation on FPGA and implementing the proposed evaluation process by using FPGA and CPU.

Chapter 4

Evaluation and Acceleration of High Accuracy Pedestrian Detection

4.1 Histograms of Oriented Gradients

This chapter first introduces the HOG pedestrian detection algorithm. Then the evaluation of HOG detection under reduced data precision is discussed. To broaden the application of this reduced precision method, another object detection method namely ACF algorithm is assessed using the same benchmarks and evaluation metrics.

The original HOG/ACF algorithm uses single-precision floating-point for all computations. Replacing the large range floating-point data with fixed-point value may potentially cause data overflow. To further increase the computation throughput, it is ideal to use the least possible number of bits for each step but the use of lower bit-width values may introduce uncertainties in the final classification result. To find the exact bit-width that can be used in fixed-point detection, this work carefully evaluates every computation step of the HOG and ACF pedestrian detection implementation in their original code. Experiments in this work has determined that 27-bit fixed-point is sufficient to maintain a similar precision as the original floating-point data representation. Both HOG and ACF detection programs are constructed that perform the original detection using fixed-point data. Then the number of bits are gradually decrease, starting from 27 bits, and compared the detection outcome with the original floating-point detection to find the least possible number of bits suitable for HOG and ACF detection. The detailed detection algorithm and experiment results at different bit-width is described in this chapter.

4.1.1 Orientation and Magnitude Computing

Input pixel values are converted to gradients in HOG-based object detection. As shown in Equation 4.1, the gradient of

$$\begin{cases} dx = pixel(x+1,y) - pixel(x-1,y) \\ dy = pixel(x,y+1) - pixel(x,y-1) \end{cases} \tag{4.1}$$

pixels, dx and dy are obtained by using a simple 1-D mask $\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$. Then, for each pair of dx and dy, the magnitude $m(x,y)$ and orientation $\theta(x,y)$ are computed as Equation 4.2.

$$\begin{cases} m(x,y) = \sqrt{dx^2 + dy^2} \\ \theta(x,y) = atan\frac{dy}{dx} \end{cases} \tag{4.2}$$

For colored images, the magnitudes are computed for each individual channel and the one with largest magnitude value is chosen.

4.1.2 Histogram Generation

In this algorithm, every 8×8 pixels form a cell, and every 2×2 cells form a block, as illustrated in Figure 4.1. The magnitudes are binned into histograms based on the orientations within each cell. Figure 4.2 shows the binning diagram used in HOG. Each cell generates a 9-bin histogram for orientation in the range of $0° - 360°$. The orientations are ”unsigned” meaning that from $180° - 360°$ the binning are the same as $0° - 180°$. The bin value is updated by weighted magnitude value. The magnitude weight is based on the difference between the angle and bin edge as shown in Equation 4.3 (floor function is used to compute bin edge).

$$\alpha = \frac{9 \cdot \theta}{\pi} - floor(\frac{9 \cdot \theta}{\pi} - 0.5) \tag{4.3}$$

In addition, the bin after current bin will also be updated to reduce aliasing as shown in Equations 4.3 and 4.4 ($vote_0$ is

$$\begin{cases} vote_0 = (1 - \alpha) \times m \\ \\ vote_1 = \alpha \times m \end{cases} \tag{4.4}$$

for current bin, $vote_1$ is for the next bin). Furthermore, each vote in a cell is bilinearly interpolated to the neighboring cell. Finally, a Gaussian filter is applied to each vote based on its location within a block to mitigate the contribution of pixels close to the block edge. Thus, the final votes can be written as the products of the vote and two weights ($weight_{intrpl}$, $weight_{gauss}$) as in Equation 4.5. Histograms within a block are

$$\begin{cases} vote_{f0} = weight_{intrpl} \times weight_{gauss} \times vote_0 \\ \\ vote_{f1} = weight_{intrpl} \times weight_{gauss} \times vote_1 \end{cases} \tag{4.5}$$

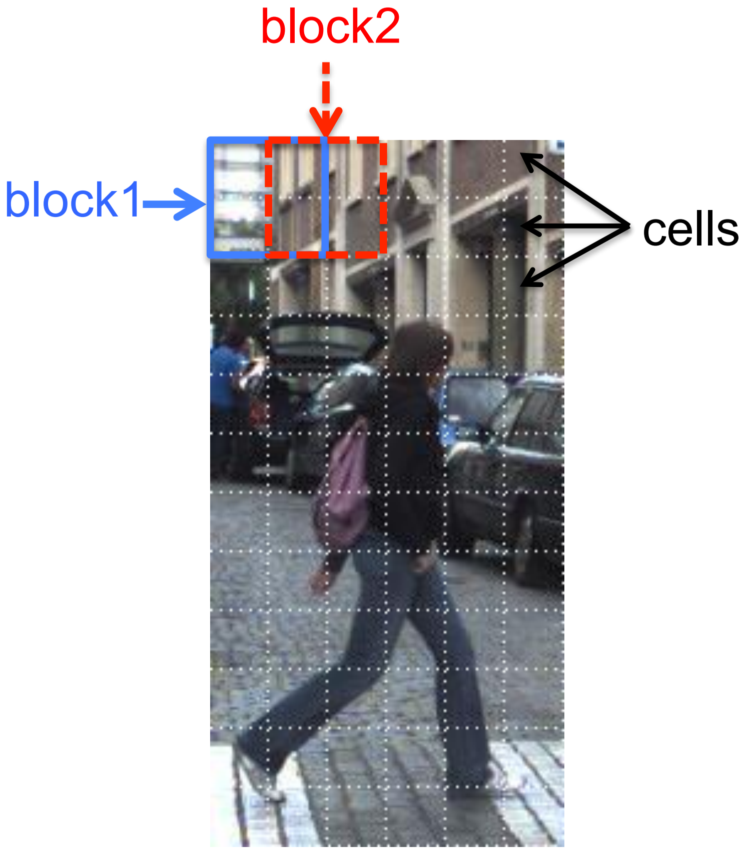

Figure 4.1: Illustration of HOG cells and blocks. A detection window consists of 6 × 12 cells of 8 × 8 pixels. Every four cells (2 × 2) are a block. Pedestrian image from [102]

concatenated together forming a 1×36 vector. All vectors in a sliding window are also concatenated as the final *descriptor vector*. Therefore, a 48 × 96-pixel window (Figure 4.1) has 5 × 11 blocks with a total of 1980 histograms (a 1 × 1980 vector).

4.1.3 Histogram Normalization and SVM Classification

Block histograms are normalized to minimize the effect of local illumination variance and foreground-background contrast. The block histogram vector is normalized twice using Equation

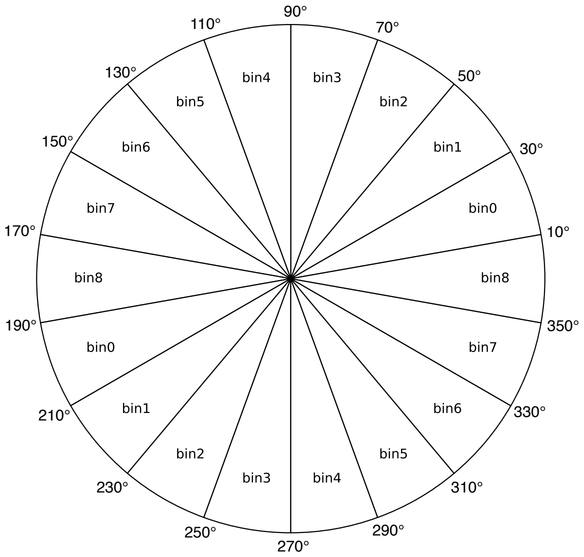

Figure 4.2: HOG cell binning. The bins are actually spaced from $0° - 180°$. Binning from $180° - 360°$ is the same as $0° - 180°$.

4.6. In general, each vector element is divided by the

$$\vec{V} = \frac{\vec{v}}{\sqrt{||\vec{v}||^2 + c}} \tag{4.6}$$

vector's Euclidean length (square root of elements' sum of squares). Constant value c is used to avoid division by zero. In the first normalization, c value is 3.6 and the maximum value for each element is limited to 0.2 after normalization. Then, the new histogram vector is normalized again using Equation 4.6 with $c = 0.001$.

Normalized histograms within a detection window are concatenated into a single vector and passed to a Gaussian kernel linear SVM classifier [17] for final classification. The SVM classifier creates a large margin around the decision boundary (hyperplane) to achieve maximum classification performance [91, 81]. Specifically, the final value s for a detection window is the dot product of the trained classification vector (normal vector to the hyperplane) $\vec{W}$ and normalized

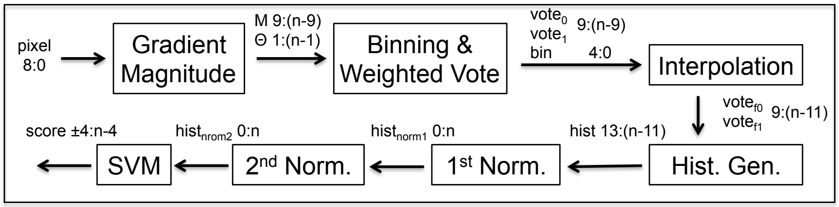

Figure 4.3: HOG computation data-flow diagram and key parameter data sizes (integer:fractional) used in this implementation.

HOG descriptor $\vec{V}$ plus a constant intercept term s_0, as shown in Equation 4.7.

$$s = \vec{W} \cdot \vec{V} + s_0 \tag{4.7}$$

The length of the SVM classifier depends on the detection window size as described in Section 4.1.2. For a 48×96-pixel window descriptor histogram vector has 1980 values (11 *vertical blocks*, 5 *horizontal blocks* and 36 *histograms/block*) and a 64×128-pixel window descriptor histogram vector has 3780 values. The final value output, s, is used to determine whether or not a window contains an object.

Figure 4.3 shows the entire data-flow of HOG detection for a single scale image. Values associated with parameters show the n-bits fixed-point implementation used in this experiment with integer and fractional sizes. All weights discussed above have 0 integer bits and n fractional bits (see Section 4.2).

4.2 Benchmark Comparison

In this section, the accuracy of the fixed-point HOG detection is evaluated and compared to the OpenCV's floating-point detection.

4.2.1 Implementation of Fixed-Point HOG Detection

For the implementation of fixed-point HOG pedestrian detection, this thesis started with the Daimler detector (a pre-trained SVM classifier) came with OpenCV [5, 26] with a window size of 48×96 pixels. The window stride is 4 pixels for both horizontal and vertical direction. Moreover, the final threshold is chose as 0.5 (only when $s > 0.5$, the window is considered as positive) to limit the total number of positive windows. All other parameters discussed in Section 4.1, *e.g.* trained classifier vector values, are converted to fixed-point data for detection.

The implementation includes all the steps of HOG detection: from the initial orientation and magnitude computation to the computing of the final score s. The final grouping algorithm (combine multiple detection windows at various scales into a single rectangle) is not included. For an n-bit fixed-point implementation, the bit-width of individual parameters are shown in Figure 4.3. As the bit-width is reduced, all intermediate values are scaled accordingly, as shown in Figure 4.3. However, the sum of histogram squares in Equation 4.6 (denominator part without computing square root) for the first normalization has a very large data range. Thus it will remain 27 bits with 0 fractional bits for n is 16 or lower. All constant parameters in HOG detection are converted to fixed-point using 0 integer bits and n fractional bits, as discussed in Section 4.1. Also the interpolation and Gaussian weights (Equation 4.5) are combined into a single value before converting to fixed-point.

4.2.2 Evaluation Methodology

Traditionally, fixed-point arithmetic implementation focuses on the absolute errors introduced by the reduced bit-width. Specifically in object detection, both fixed-point and floating-point object detectors are applied to detection windows known as object or background for detection rate comparison. The desired bit-width is determined by the minimum acceptable detection rate using certain fixed-point bit-width. However, this approach may not correctly predict the actual detection performance when considering the entire frame across multiple image scales. Usually a post-processing step is performed on all positive windows across the image at all scales to merge nearby positive windows. This step can reduce the number of false positive windows found by the detector. On the other hand, it can introduce detection errors such as incorrectly detected object sizes that would otherwise not have been found in window-based evaluation. Thus, to evaluate the effect of reduced data precision, methods other than window-based evaluation are needed. Dollar *et al.* [20, 21] proposed the *per-image* evaluation approach as opposed to *per-window* methodology for pedestrian detection algorithm evaluation. They reported the classification performance of various classifiers for these two approaches. In general, the *per-image* based approach is more meaningful as well as practical. Therefore, this method is applied to the fixed-point detection to find the optimal bit-width. The detailed evaluation results will be discussed in Section 4.2.4 .

4.2.3 Benchmarks and Detection Evaluation

Four benchmarks are used to evaluate the fixed-point HOG detection: Daimler Mono Pedestrian Detection [26], TUD-Brussels [102], Caltech Pedestrian Detection [20], and three sequences from ETH datasets (the BAHNHOF, JELMOLI, and SUNNY DAY sequences) [27]. All benchmark images are 640×480 and have a ground truth of pedestrians. Every ground truth pedestrian is marked by a rectangular bounding box (BB), indicating its location and size. The evaluation only selects the frames that contains at least one pedestrian object with a BB height > 67 pixels

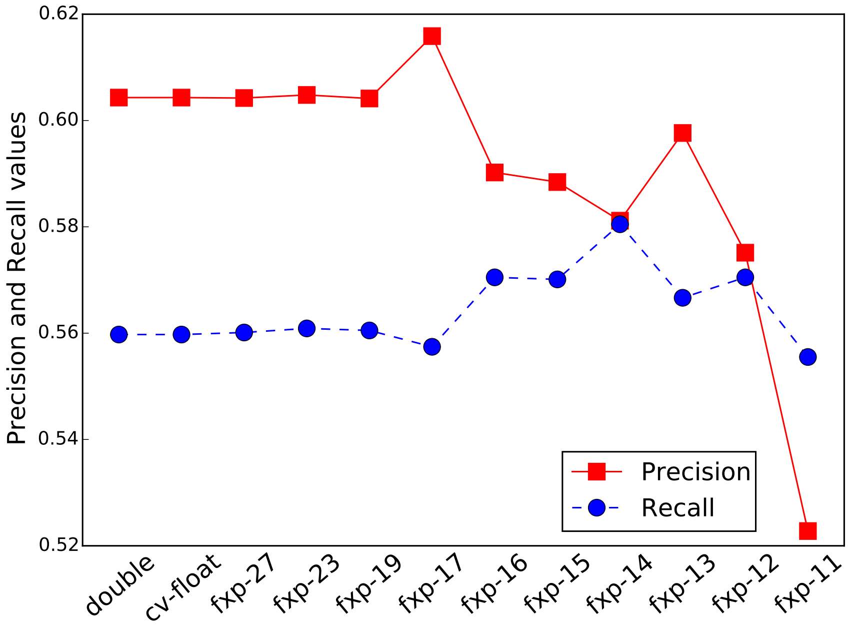

Figure 4.4: HOG fixed-point detection results for Daimler Benchmark.

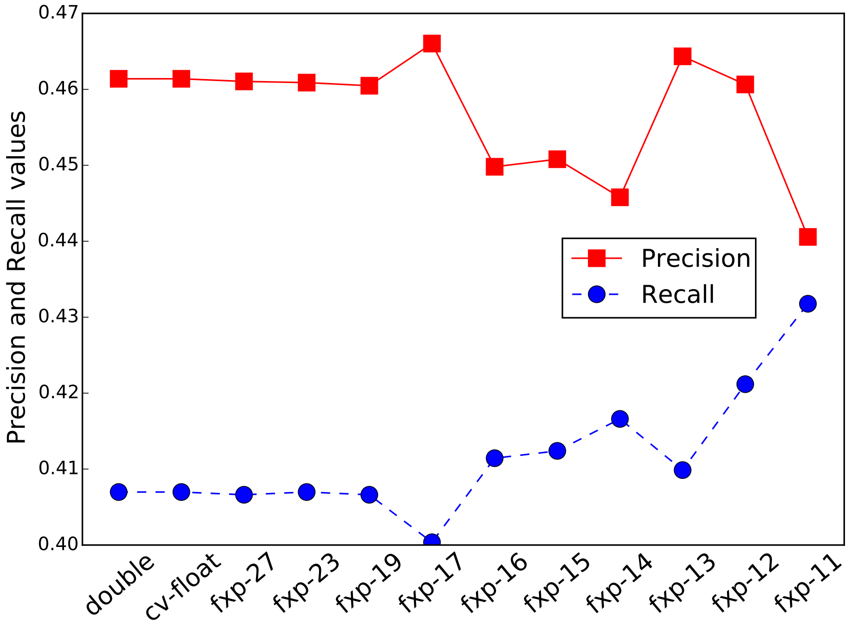

Figure 4.5: HOG fixed-point detection results for Caltech Benchmark.

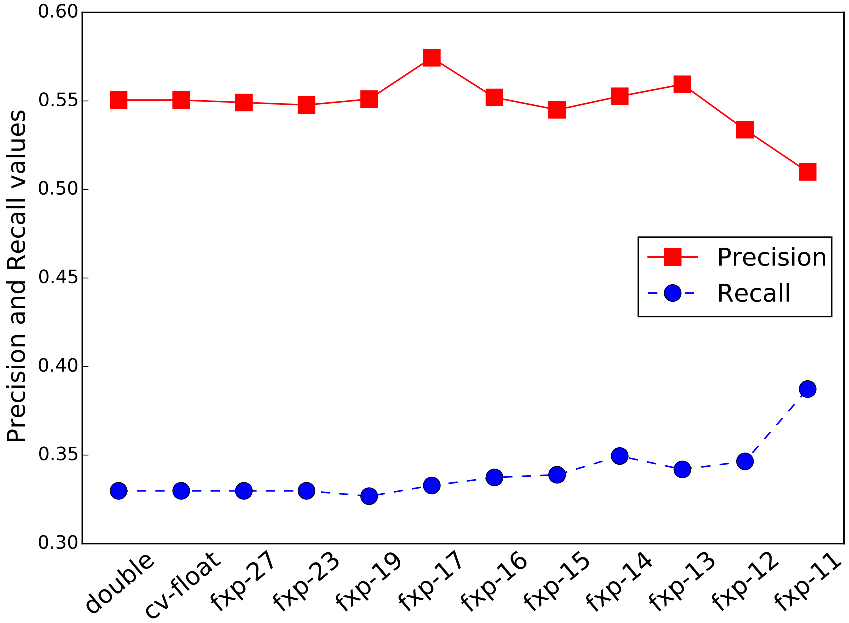

Figure 4.6: HOG fixed-point detection results for TUD-Brussels Benchmark.

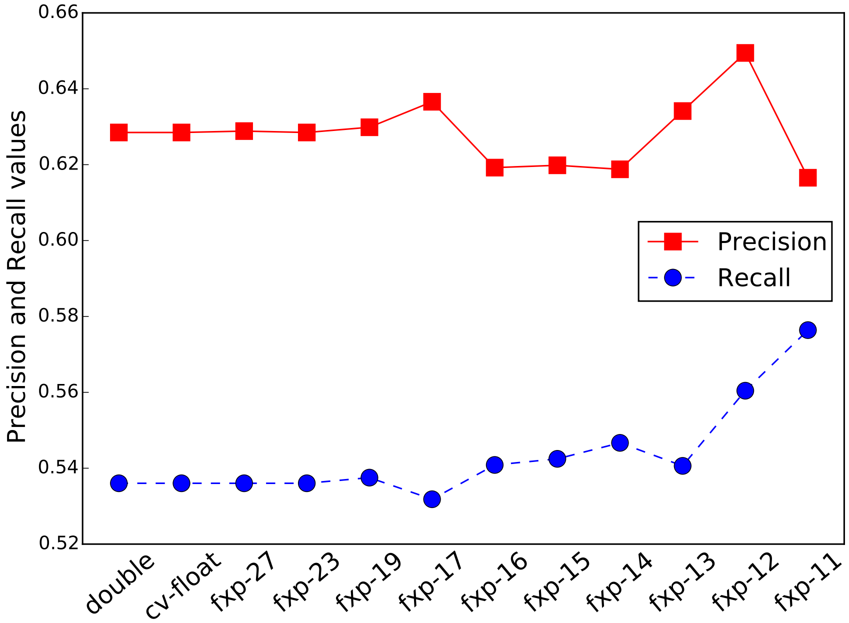

Figure 4.7: HOG fixed-point detection results for ETH Benchmark.

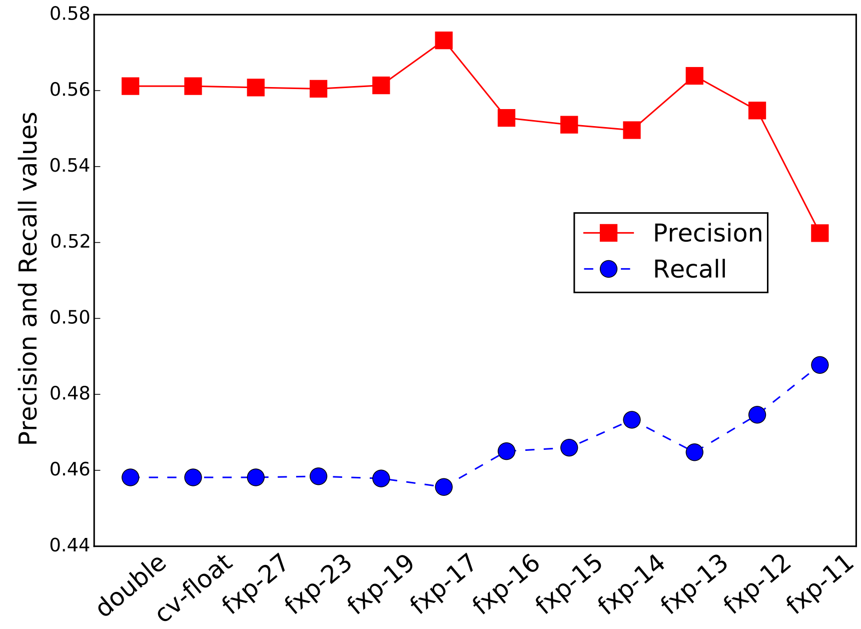

Figure 4.8: HOG fixed-point detection results for all Benchmark.

(70% of the detection window height). The number of images and pedestrian objects used for evaluation are shown in Table 4.1. To match the detection results to the ground truth, the commonly accepted PASCAL method is used as shown in Equation 4.8 [28]. BB_{det} refers to the BB from the detection and BB_{grt}

$$\alpha_0 = \frac{area(BB_{det} \cap BB_{grt})}{area(BB_{det} \cup BB_{grt})} > 0.5 \tag{4.8}$$

is the ground truth BB. Two objects are matched when their overlapping area is more than 50% of the union area. Each detected object may be matched at most once to a ground truth object. In addition, adjustments for both BB_{det} and BB_{grt} are made based on the methods described in [20, 21]. For each BB_{grt}, the aspect ratio of a rectangle depends on the limb position of a walking pedestrian. Thus, all BB_{grt} are resized to an aspect ratio of 0.41 by keeping the center of the object. What's more, each BB_{det} corresponds to a detection window of 48×96 pixels with about twelve-

Table 4.1: Number of frames and objects for each benchmark sequence after filtering.

	Daimler	Caltech	TUD-Brussels	ETH
# image	2117	5346	237	1785
# object	2603	8310	661	8076

pixel paddings on top and bottom of each pedestrian [26]. Therefore, the BB_{det} height is resized by a scale of 0.78125, then the aspect ratio is resized to 0.41. These processes provide better matching between ground truth and detection result. Moreover, ground truth objects near the image edge, with height below 67 pixels and non-pedestrian are set to ignore. Ignored objects are not counted as true positive if matched and will not contribute to false negatives if unmatched.

4.2.4 Evaluation Result

The fixed-point detection in this work performs detection in 27-bits down to 11-bits. The number of bits for each fixed-point detection is shown in Figure 4.3 (substitute n with corresponding bits). In addition to the single-precision floating-point and fixed-point detection, another detection is constructed with all data represented by double-precision floating-point. All detection results are collected, and evaluated using the method discussed above. Then the precision and recall are calculated from number of true positives (TP), false negatives (FN), and false positives (FP). Finally, all results discussed below are using for *per-image* evaluation with each 640×480-pixel image processed at 34 different scales with a scale factor of 1.05.

Result shows the OpenCV detection (*cv-float*), double precision floating-point (*double*) and a subset of fixed-point detection results for each benchmark and the overall results by averaging the four individual benchmark, in Figure 4.8. Detection precision and recall are computed using Equation 4.9.

$$\begin{cases} precision = \frac{TP}{TP+FP} \\ recall = \frac{TP}{TP+FN} \end{cases} \tag{4.9}$$

Fixed-point detection results from 27-bits to 18-bits are almost identical to the floating-point results in all benchmarks. For 17-bits and lower, detection at lower bits generally increases recall and decreases precision. However the precision and recall for BANHNOF sequence at *fxp-12* and *fxp-13* are both increased. This results in precision increase in ETH benchmark at *fxp-12* and *fxp-13* and thus the overall performance increase as shown in Figure 4.8. For all benchmarks, it is observed an increase of precision in *fxp-17*, and a decrease in recall. This is due to a slight decline of the TP, but a significant reduction in FP. Besides, the reduced TP also results the contraction of FN, hence the loss of recall as shown in Figure 4.4, 4.5, 4.7 for *fxp-17*. The overall recall is increased from 0.458 at *cv-float* to 0.465 for *fxp-13* and 0.475 for *fxp-12* while the precision grows from *cv-float*'s 0.561 to 0.564 for *fxp-13* and dropped to 0.555 at *fxp-12*. Moreover *fxp-11* has boosted recall to 0.488 with a significant decrease of precision to 0.522. Finally, 13-bits is chosen as hardis chosen asplementation as it provides a balance between precision and recall and consistent performance across all benchmarks in per-image evaluation. The detailed hardware implementation is discussed in Section 4.3.

4.2.5 Aggregate Channel Feature (ACF) and Its Fixed-point Evaluation

ACF algorithm is an extension of the popular HOG algorithm that incorporates multiple feature types [22]. As a result, it shares some of the processing steps as HOG such as gradient computation and histogram building. In ACF algorithm, pixel values, pixel gradients and HOG form 10 channels of features (three channels for color, one channel for gradients, and six channels for HOG) to describe an object. The pixels value features and all following computations are perform in CIELUV color space. To obtain CIELUV color space pixel values, the pixels in the original RGB format is first converted to CIEXYZ color space using a linear transformation and then converted to CIELUV color [42]. From the cCIELUV pixels, gradient magnitudes are computed using the same mask in HOG algorithm shown in Equation 4.1. Unlike HOG that uses 9-bin histogram, ACF uses a 6-bin histogram ($[0°, 180°]$ range) for every non-overlapping 4×4 block (bi-linear interpolation is also applied here). The color and gradient magnitudes are also divided into the same 4×4 blocks by summing up all values inside a block. After all 10 channels are generated, each channel is normalized by a $[121]/4$ kernel independently to obtain the finally ACF features. Similar to HOG algorithms ACF shrinks the original image at multiple scales to capture pedestrians with different sizes. Note that in this algorithm, all images are first up-sampled to 1280×960 (used as the initial scale) before perform feature extraction to obtain best performance. For 1280×960 image, there are 27 scales per image. Unlike HOG algorithm, the sliding window size used in this ACF study is 64×128-pixels as the HOG algorithm. However, since the pixels are summed up in the in the channels, the window size becomes 16×32 in the channel map (and the window stride is 4 pixels in feature map). To determine if a window is a candidate object in the multi-channel feature map, an AdaBoost classifier is used [29]. The classifier used in this work is trained based on the original

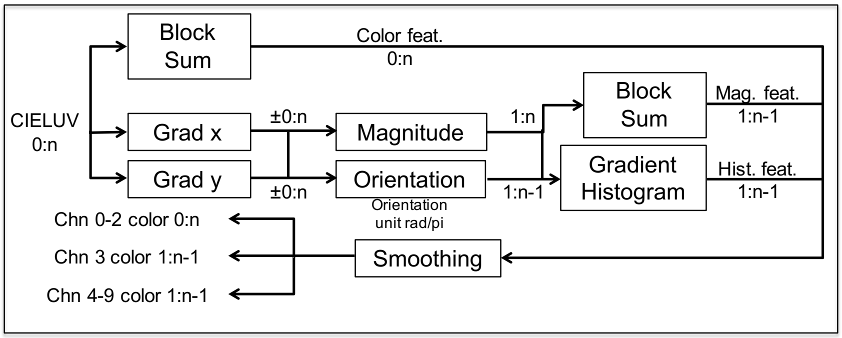

Figure 4.9: ACF computation data-flow diagram and key parameter data sizes (integer:fractional) used in this thesis.

code in [22] using INRIA pedestrian dataset [17]. A non-maximum suppression method is applied to the final positive windows to minimize the duplicate windows [22].

To evaluate the reduced-precision effect on the outcome of ACF detection, a fixed-point detection program is constructed. For n-bit fixed-point detection, the bit-width used in each step of the feature extraction is described in Figure 4.9. Note that in this work, it is assumed that colors are alreadyconverted to CIELUV color space (and normalized to $[0, 1[$ range) before performing fixed-point detection. The final classifier thresholds is converted to the corresponding bit-width as the features (for color channels, the fractional bit-width is different from other channels)

The evaluation metrics for the fixed-point ACF algorithm is the same as the HOG evaluation. Figures 4.10, 4.11, 4.12 shows the evaluation of the three pedestrian detection benchmarks (Caltech, TUD-Brussels, and ETH). Note that the Daimler benchmark is not used in this evaluation as ACF algorithm requires color images instead the gray-scale image used in Daimler benchmark. Similar to HOG fixed-point evaluation, the detection under reduced precision shows significant in-

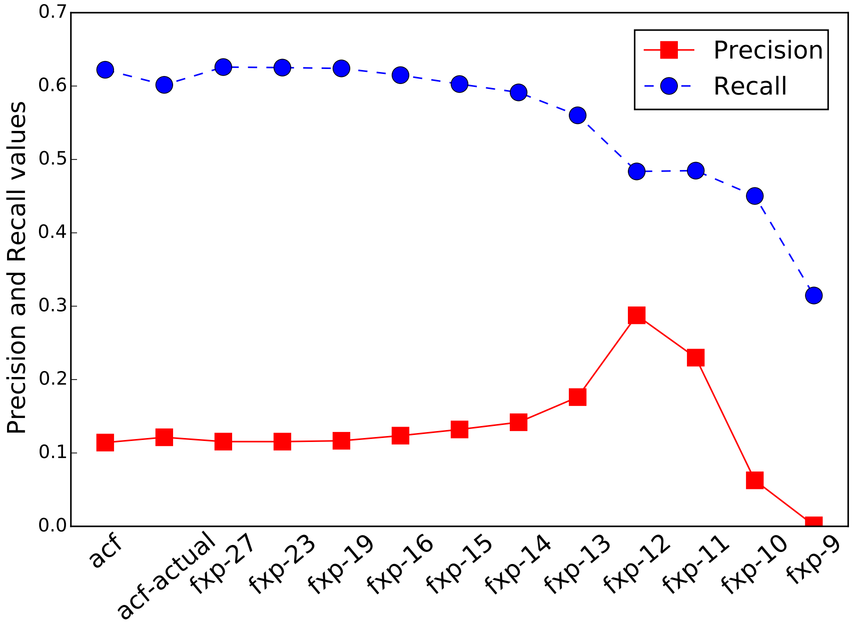

Figure 4.10: ACF fixed-point detection results for Caltech Benchmark.

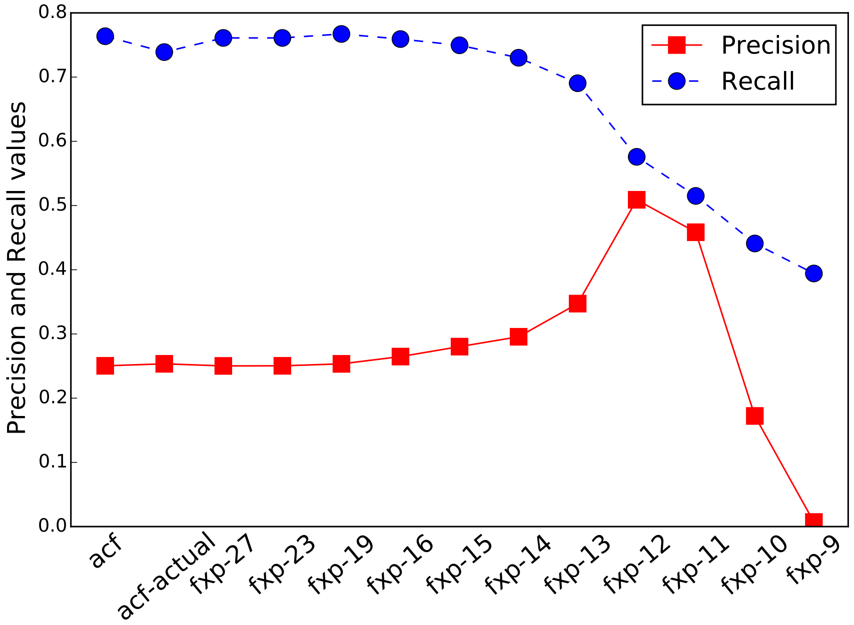

Figure 4.11: ACF fixed-point detection results for TUD-Brussels Benchmark.

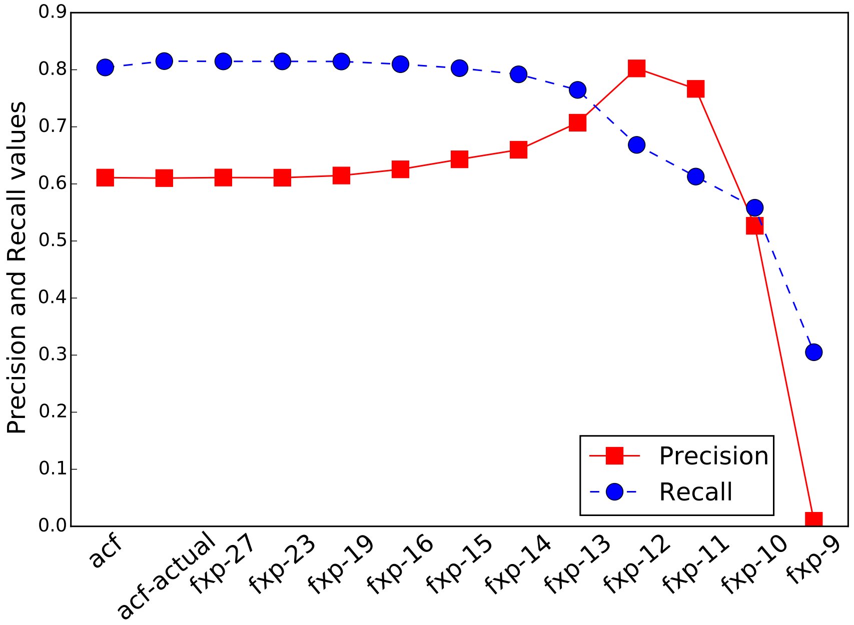

Figure 4.12: ACF fixed-point detection results for ETH Benchmark.

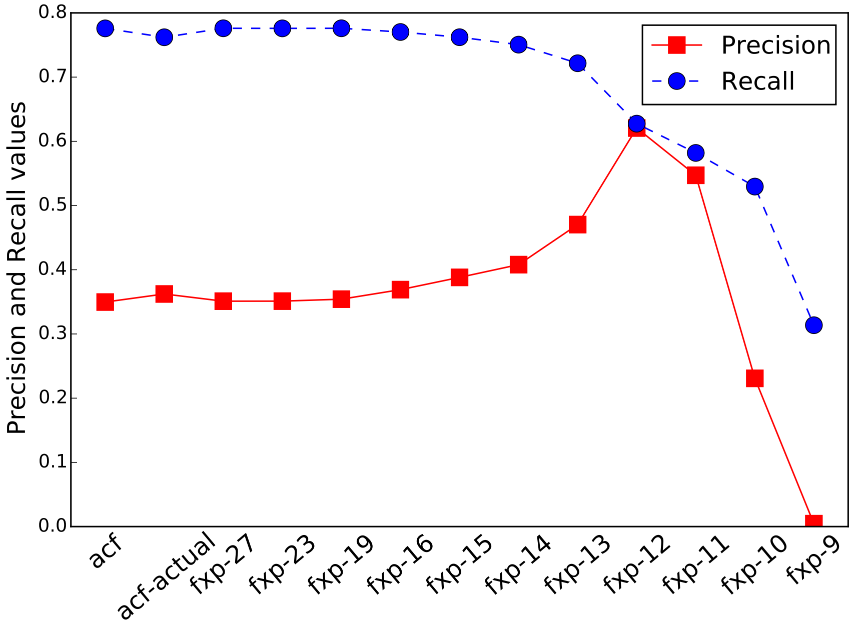

Figure 4.13: ACF fixed-point detection results for all Benchmark.

crease in precision when the bit-width has been increased. On the other hand, the recall drops due to the growth of false positive objects detected under low bit-width. This result is consistent with the HOG detection. Thus, the method of reducing bit-width for feature extraction and classification can potentially be extended to broad range of applications.

4.3 FPGA Implementation

This section describes and evaluates the fixed and floating-point implementations of the HOG detection on an FPGA. The throughput of the FPGA implementation is compared to those of the CPU and GPU implementations.

4.3.1 FPGA Platform

The HOG detection system is implemented on the Convey HC-2ex machine [2]. The system's hybrid-core architecture is composed of two Intel Xeon E5-2643 four-core processors and four Xilinx Virtex-6 LX760 FPGAs. Both CPUs and FPGAs share a globally addressable 256 GB virtual memory, 128 GB on FPGA side and 128 GB on CPU side. FPGA memory is connected to CPUs via one PCIe 3.0 x16. The FPGA memory system is built around Convey's Scatter-Gather DIMMs to provide random transfer of 8-byte bursts at near peak bandwidth [8]. All FPGAs are linked to host processors through an Application Engine Hub that can send and transfer opcodes and scalar operands to FPGA. Each FPGA has 16 64-bit memory channels at 150 MHz controlled by eight memory controllers. The FPGA program runs at 150 MHz. The memory subsystem provides a highly parallel and high bandwidth (19.2 GB/s per FPGA) connection between FPGAs and physical memory. These properties permit the user to design complicated memory access pattern in the

HOG-Engine to achieve maximum performance. The hardware architecture and memory accesses will be discussed in the following sections.

The host software is written in C++ and the FPGA code is developed in Verilog. The design is simulated using Convey Personality Development Kit and Modelsim Foreign Language Interface for hardware and software co-simulation. Synthesis is performed using Xilinx ISE 14.3. Xilinx Core Generator is used to generate fixed-point multiplication and square root IP cores. For fixed-point division, the divider from [89] is used. Each HOG-Engine uses 138 fixed-point multiplication modules. To ease the FPGA timing, 64 multiplication modules in bi-linear interpolation of votes and four multiplication for magnitude voting are implemented on DSPs (a total of 68 DSP slices per HOG-Engine), all others are on LUTs. The normalization module is implemented by using square root, division and multiplication modules. First the histogram squares are summed, then sent to square root module. Finally the reciprocal is computed by the divider core. The normalized histogram value is the multiplication of histogram and the reciprocal value.

4.3.2 HOG-Engine Architecture

As a first step, the HOG pedestrian detection code is profiled on CPU, to find the most critical computation in HOG detection. The profiling information is shown in Table 4.2. Post processing is used in all object detection algorithms to combine similar windows into one. The HOG-Engine is focused on implementing the most computational expensive parts of HOG detection on FPGA: orientation binning, magnitude voting, histogram generation, normalization, and SVM classification. All other computations are performed in software.

This implementation design on FPGA consists of two steps: histogram generation and classification. Histogram generation produces weighted votes, accumulates them in cell histograms

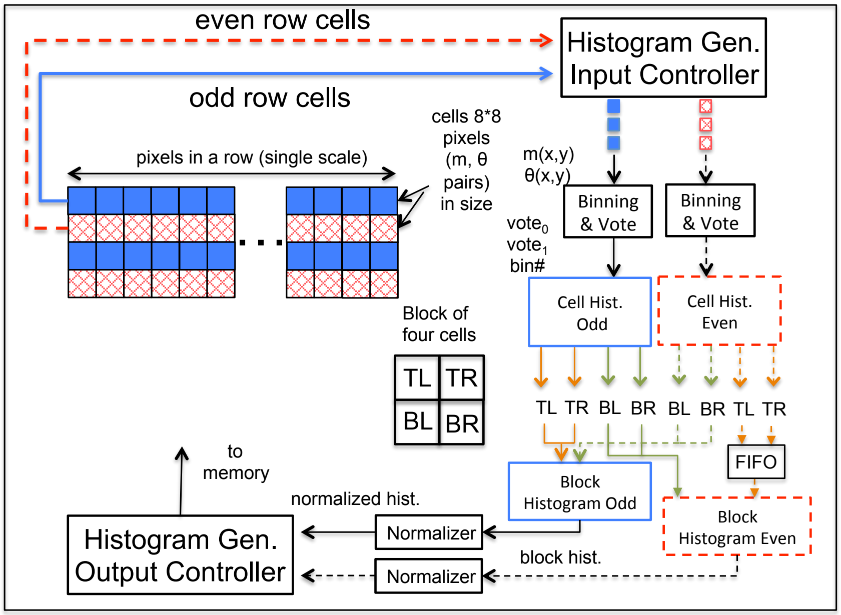

Figure 4.14: The HOG-Engine histogram generation architecture. Two rows of cells are processed in parallel. A single MC will fetch pixels in odd and even rows periodically. See text for details.

that are combined to form block histograms. Block histograms are then normalized twice and sent back to memory. The classification module fetches the normalized histograms from memory and performs SVM classification to generate the final score. The schematics of the HOG-Engine for histogram generation and classification are shown in Figure 4.14, 4.15 respectively.

The HOG-Engine reads the magnitude and orientation values from memory. Each pair of magnitude and orientation values is packed into a single 32-bit integer. As one memory access returns a 64-bit value, two pairs of magnitude and orientation values are returned in a single memory access. The HOG-Engine fetches pixels from two rows of cells alternately to increase parallelism as shown in Figure 4.14. For each pair of orientation and magnitude, two vote values ($vote_0$, $vote_1$) and a bin number are computed. As discussed in Section 4.1.2, each cell has 64 pixels and generates a 9-bin histogram. However, due to bilinear interpolation, each vote is weighted and interpolated

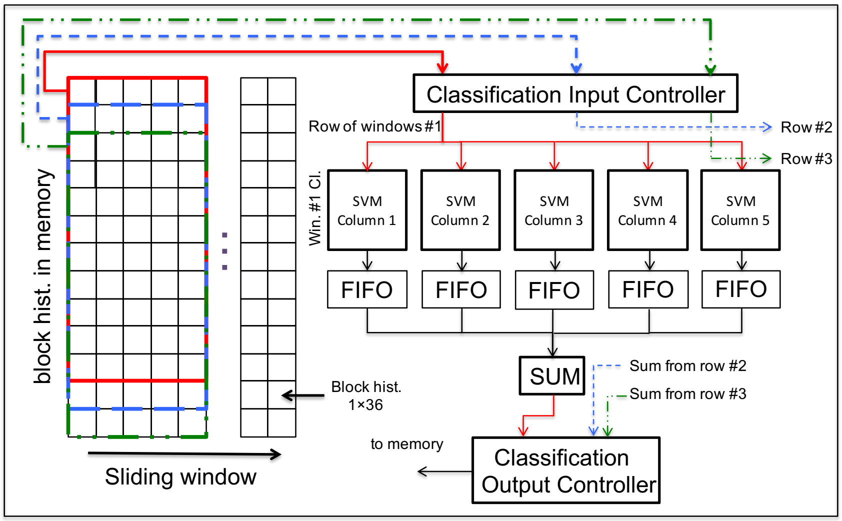

Figure 4.15: The HOG-Engine classification architecture. Each copy of classification module will compute svm for a column vector at 5 different positions. When five column vectors in a row are processed, a sum for the detection window will be produced.

into all other cells in the same block. To reduce the interaction between different cells, each cell produces 4×9-bin histograms. In addition, as a cell could be in one of the four positions in a block shown in Figure 4.14 (TL, TR, BL, BR), in this implementation, a single cell will generate 4×36-histograms. Cell histograms are then combined using a simple vector add to obtain the block histogram when all four cells in a block are processed. Unused cell histograms are automatically discarded based on their position (cells on the edge of image can only appear in a subset of positions of a block). Besides, to compute the block histogram between the second and third rows, the *TL, TR* histograms of second row cells are saved in *FIFO* and later combined with the third row's *BL, BR*. Two Block histogram generation modules are instantiated for each HOG-Engine. Generated block histograms are sent to normalizer and finally stored in memory. This design is efficient as all pixel values within an image scale are accessed only once to generate the histogram.

Table 4.2: HOG detection profiling result.

Function	Time (%)
Initialization & read image	0.65
Image resize	0.65
Magnitude& angle	0.62
Binning & voting	3.44
Block hist. gen. & norm.	46.20
SVM	18.82
Post processing	29.63

Generated histogram values are stored in memory, to be processed by a sliding-window based classification system (window stride is one block). In this classification system, detection windows are computed in column basis. As the detection window is slid one block to the right, only one new column of block histograms are fetched from memory. This work divides, therefore, the classifier into five classifiers, one for each column as shown in Figure 4.15. Every classifier performs part of the SVM classification and output a single value for the 396 elements. Each column of blocks will be sent to all five classifier as they will be at five different locations when the window slides in a row. The *Sum* module adds these values together for each window and outputs the final threshold sum. When the window slides down one block, all the block histograms within that window are re-fetched from memory. As result, the classification is less efficient as the histogram generation.

In this design,three classification modules are used to compute three consecutive row of windows so the speed of classification is similar to histogram generation.

The aforementioned architecture works well with a window stride of *eight* pixels (one cell). However, to further improve the detection accuracy, a stride of *four* pixels (half cell) is used. When the window stride is four pixels, all cells and blocks in the new window are changed and previously computed cell histogram results connot be reused. To solve this problem, a single scale of image is treated as four sub-scales, processing each with a window stride of eight pixels. Concretely, the sub-scale starting at the first column, first row of pixels is firstly processed using the above HOG-Engine. Then, the same image is processed again starting at the first row, fifth column. Thirdly, histograms starting at the fifth row, first column, are computed and finally fifth row, fifth column. Therefore, a total of 34 scales image is divided into 134 sub-scales (the last scale only have two sub-scales). This design allows us to use the same HOG-Engine architecture to efficiently generate histograms and perform classification.

4.3.3 Input/Output Controller

Both histogram generation and classification modules have input and output controllers to interface with the FPGA memory system. As discussed previously, the HOG-Engine processes a frame at 34 different scales. In addition, each scale is divided into four sub-scales to slide detection window by four pixels vertically and horizontally. These controllers are responsible to access images at different scales. The image is resized in software and then magnitude and orientation are computed. For a single frame, magnitude and orientation values at 34 scales are concatenated into a single array and sent to FPGA memory. Histogram generation input controller generates pixel (magnitude and orientation pairs) addresses by using three nested state machines, to control horizontal cell offset, vertical cell offset and pixel offset within a cell. The image size information such as the number of horizontal cells, vertical cells, offset to current scale, and sub-scale are stored in ROMs. These offsets are added together with image base address to form the actual pixel address. A counter is used to keep track of current sub-scale number and incremented when all addresses in that scale are generated to control the output of ROMs. As a result, no DSP slices are needed in input/output controllers. The input controller for the classification system operates similarly, but generates three addresses in parallel for three rows of detection windows as discussed in 4.3.2. Besides counting the number of scales processed, the output controllers also count the number of histograms/final scores processed for all scales to determine the ending-point of a frame. Each HOG-Engine has a dedicated memory channel for histogram output but three HOG-Engines on one FPGA share a single memory channel for final score output through time multiplexing. Since the number of output values in the final classification are 52x less than the input pixels, multiplexing three outputs into one memory channel will not affect the system throughput. Synchronization between the histogram generation system and classification system are done by a simple 1-bit FIFO. When the histogram output controller finished one scale, it writes one 1-bit value into the FIFO to indicate data available for classification. The classification input controller will read one value out when finished a scale to prevent the FIFO full. The histogram input controller will stop working when the 1-bit FIFO is full (all memory allocated for histograms are used).

To allow maximum throughput for the FPGA execution, the HOG-Engine execution at multiple scales/frames are also pipelined. Specifically, after finished fetching pixels at one size, input controller modules will immediately start the next scale/frame, if available. The HOG-Engine operates without knowing the size of the image. However, the histogram generation module needs to know the beginning and ending of each column and each row to combine the cell histograms to block histogram and discard unused values as noted in Section 4.3.2. What's more, since the HOG-Engine operates on two row of cells, the last row will have only one module working if the image has odd number of cell rows. Therefore, the histogram input controller generates a four-bit position signal associated with each pixel to let the core know which portion of image it's currently executing on. Two-bits indicate the beginning, middle and end of a column and the other two bits used for row. The same idea is also applied to the classification module as the first four columns and last columns will not be sent to all five SVM classifiers. By changing the ROMs containing the image size information and the constant scale number in the input and output controllers, the FPGA implementation can be used for any image sizes and scale factors. As a result, this design is highly scalable.

4.3.4 FPGA Resource Usage Comparison

This section reports the area utilization and clock frequency of the HOG-Engine. The data is shown in Figure 4.16 for fixed-point, 27 to 13 bits, and single-precision floating-point. The resources usage does not include input and output controllers that interface with external memory. FPGA resources for the actual implementation including all functional units will be discussed later. Percentage values are based on the Xilinx Virtex-6 LX760 FPGA. Compared to floating-point, the registers used for fxp-13 are reduced by a factor of 3.0x, LUTs by 6.6x, DSPs by 2.6, BRAMs by 2.2, and frequency increased by 3.1x. The floating-point implementation is fully pipelined with maximum number of pipeline stages applied to most arithmetic operations (generated by Xilinx Core Generator), except the adder at the histogram accumulation. Since the design is fully pipelined, a new vote will be accumulated to the existing bin value every clock cycle. Therefore, a multiple

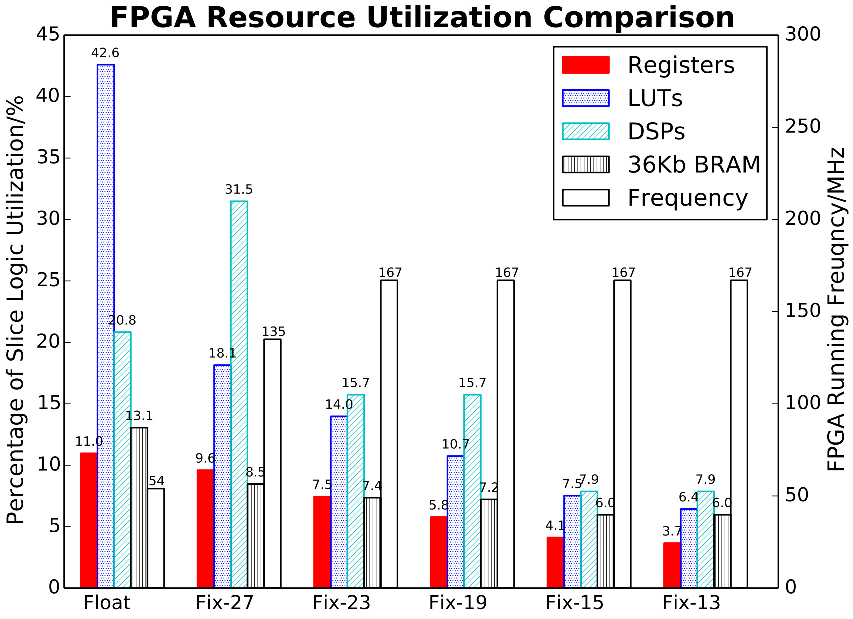

Figure 4.16: HOG-Engine FPGA resource utilization and running speed comparison. Percentage values are based on an Xilinx Virtex-6 LX760 FPGAs. The number of 36kb BRAMs also include 18kb BRAMs. See text for detailed analysis.

staged floating-point adder can not be used, hence negatively impacts the clock frequency in the floating-point implementation.

4.4 Results and Discussions

In this section, the FPGA program performance is evaluated and compared with CPU, GPU, and FPGA. All the tests are based on 640×480 images with a scale factor of 1.05. To the best of our knowledge, this is the first densely scanned detection window implementation of HOG algorithm on FPGA. The window stride is four pixels for both directions. Therefore, for each frame, there are 134 scales with window stride of eight as noted in Section 4.3.2.

Table 4.3: HOG-Engine complete system resource utilization. Three HOG-Engines are instantiated on a single FPGA using all 16 memory channels.

Resources	Registers	LUTs	36Kb BRAMs	DSPs
Percentage	22%	39%	53%	22%

4.4.1 FPGA Execution Speedup

As discussed in Section 4.3.1, this FPGA implementation is targeted on Convey HC-2ex computer. Each FPGA is running at 150MHz clock with 16 64-bit memory channels at the same frequency. Three instances of HOG-Engine are implemented on a single FPGA taking all 16 memory channels. The complete system resource utilization, including three HOG-Engine, input/output controllers, and Convey wrapper, is shown in Table 4.3.

As the program is fully pipelined across different image scales, the total execution speed is determined by the number of memory accesses. The experiments are performed on the Convey HC-2ex computer using a single FPGA by measuring only the FPGA execution time. Memory copy time is not taken into account since this latency can be hidden by pipelined execution. The experiment with single HOG-Engine indicates the FPGA can process one image at all 34 scales in 44 ms. With three HOG-Engines executing in parallel, this work is able to achieve an overall throughput of 68.2 fps on a single FPGA.

For floating-point implementation, only one engine can be placed on a single FPGA with a reduced clock frequency and requires eight input memory channels and four output memory channels for each HOG-Engine. Thus, the speed is estimated to be 8.79 fps if under full memory bandwidth (see Table 4.5, FPGA-float). Hence, the fixed-point implementation has increased the throughput of the FPGA execution by at least 7.8x. Moreover, it is projected that the speed of executing the fixed-point HOG-Engine on all four FPGAs is 273 fps. In this design, the magnitude and orientation array size of a single image (34 scales) is 12.6 MB, and the size of FPGA output array (final scores) for an image is 0.24 MB. Running at 273 fps requires 3.5 GB/s memory bandwidth which is well below the bandwidth of 15.75 GB/s delivered by the 16x PCIe 3.0.

Based on the single FPGA execution speed, the processing speed for larger images is also estimated by scaling the throughput based on the number of memory accesses, as shown in Table 4.4. The number of memory read requests are the input requests for the histogram generation module. Since the design is fully pipelined, the throughput is determined by the memory bandwidth. This estimation can correctly predict the execution speed for different image sizes. As seen in Table 4.4, when the original image size increased by 1.2 times, the number of read requests for the histogram generation grows by 1.6 times. This significant growth is because more image scales are needed to evaluate a single frame.

4.4.2 Speedup Comparison

To compare the FPGA implementation with other platforms, HOG pedestrian detection is performed on both CPU and GPU. The CPU and GPU implementations are all in single-precision floating-point, adapted from the commonly used OpenCV library [5] to use the parameters that matches the FPGA execution. CPU program is implemented in C++ and compiled by G++ 4.3.6. The CPU platform has two Intel Xeon E5520 quad cores with 24 GB memory. The GPU used is the Nvidia Tesla K20 GPU attached to the same machine. The results of using Intel's IPP library are also included for CPU's multi-threading capability. All execution time are measured corresponding

Table 4.4: HOG detection throughput estimation for different sized images. The throughput for image sizes other than 480×640 are estimated based on the number of read requests in the histogram generation module.

Resolution	scales	read requests	Input Size (MB)	det. wind.	FPS
$640 * 480$	34	6219520	12.6	121210	68.18
$800 * 600$	38	9906944	20.0	211788	42.80
$1024 * 768$	43	16742272	33.7	389186	25.33
$1280 * 960$	48	25915328	52.1	637332	16.36
$1600 * 1200$	52	40731520	81.8	1049886	10.41

to the portions that are implemented on FPGA. In addition, for GPU execution, the memory transfer time is not included. The throughput for all platforms is shown in Table 4.5. The single FPGA version achieves a 68.7x speedup compared to the single core CPU and a 5.1x speedup compared to GPU. If all four FPGAs are used for execution,the throughput can be pushed to 273 fps with a 20x speedup to GPU. As a result, the proposed HOG frame work is suitable for applications that require large throughput and high accuracy pedestrian detection.

4.4.3 Power Consumption Comparison

The power consumption estimation for the three platforms is shown in Table 4.6. The maximum Thermal Design Power of Intel Xeon E5520 processor is used for the CPU power measurement. For the GPU implementation, the Nvidia Tesla K20 board power is used since no individual chip power is available. FPGA power consumption, both fixed-point and floating-point, are

Table 4.5: HOG detection throughput comparison.

Platform	Throughput (fps)	Speedup
CPU	0.99	1.00
CPU-IPP	1.14	1.15
FPGA-float	8.79	8.86
GPU	13.40	13.50
one FPGA-fxp13	68.18	68.69
four FPGA-fxp13	272.73	274.77

Table 4.6: HOG power consumption comparison.

Platform	Frame/s	Power (W)	Joules/frame
CPU	0.99	80	81
CPU-IPP	1.14	80	70
FPGA-float	8.79	36	4
GPU	13.40	225	17
FPGA-fxp13	68.18	37	0.54

estimated using Xilinx Power Estimator 14.3 with a 100% toggle rate (assume all signals will flip every clock cycle). The floating-point module has less power than the fixed-point version. This is due to reduced clock frequency and less resources, since floating-point version only has single-engine running at significantly lower frequency. Then the power divided by throughput (energy

Table 4.7: Comparison of parameters and performance for various FPGA implementation.

	Scale	# scales	# bins	Win. stride	Win./frame	Resolution	FPS
FPGA Implementation							
This work	1.05	34	9	4	121,210	640 × 480	68.18
[31]	-	18	9	8	>27,960	1920 × 1080	64[1]
[12]	1.2	13	9	8	20,868	1024 × 768	13
[68]	-	1	9	8	5,580	800 × 600	72
[69]	-	1	8	9	1,540	640 × 480	62.5
[9]	-	1	9	-	1,000	800 × 600	>10
[43]	1.2	>1	8	4	56,466	640 × 480	30[1]
[34]	1.2	>1	8	4	3,615	320 × 240	38
GPU Implementation							
[34, 5]	1.05	37	9	8	unkn	1024 × 768	17
[87]	1.05	>1	unkn	unkn	4096	640 × 480	57
[66]	1.1	>1	8	2	50000	640 × 480	23.8
[104]	-	1	9	8	unkn	640 × 480	32
[11]	1.05	>1	9	4	150000	1280 × 960	2.4
[77]	1.05	>1	9	unkn	unkn	640 × 480	5.6

consumption to process a single frame, Joules/Frame) is computed as a measure of power efficiency. The fixed-point implementation uses 130x less energy than CPU and 31x less energy than GPU to process a single frame.

[1] Simulation estimated speed, no actual implementation.

4.4.4 Comparison with State-of-the-Art

Table 4.7 provides a comparison of this FPGA implementation with state-of-the-art FPAG and GPU implementations. As noted previously, this work performs HOG pedestrian detection with a densely scanned detection window that achieves very good detection result. This choice also results in increased computational complexity. Therefore in addition to the comparison of this FPGA implementation with various platforms, Table 4.7 compares the important parameters [17, 22] that determine the accuracy, speed and performance of HOG detection of previous hardware acceleration approaches with this work. As shown in Table 4.7, the work presented in this thesis is significantly faster than all previous GPU/FPGA implementations despite the number of detection windows are at least an order of magnitude higher.

4.5 Conclusion

Object and person detections are computationally intensive applications whose importance has been steadily growing. The Histogram of Oriented Gradients (HOG), one of the most popular detection algorithms, achieves a high detection accuracy but delivers just under *one* frame-per-second (fps) on a high-end CPU. All current fixed-point FPGA implementations use large bit-width to maintain detection accuracy, or perform poorly with reduced data precision. This chapter explores the FPGA implementation of HOG using reduced bit-width fixed-point representation to lessen the required area resources on the FPGA, increase the clock frequency and hence the throughput per device. The detection accuracy of the fixed-point HOG is checked by the state-of-the-art computer vision pedestrian detection evaluation metrics and show it performs as well as the original floating-point code from OpenCV. To further validate the evaluation result, the reduced precision processing is also assessed on another pedestrian algorithm ACF. The result for the ACF algorithm is consistent with HOG. Then this thesis shows that the FPGA implementation achieves a 68.7x higher throughput than a high-end CPU, 5.1x higher than a high-end GPU, and 7.8x higher than the same implementation using floating-point on the same FPGA. Power consumption estimation shows that FPGA uses 130x less energy than CPU and 31x less than GPU to process a single image. The future work involves performance comparison of different detection classifiers under reduced bit-width using the same HOG feature.

Chapter 5

Optimizing Hardware Design for Human Action Recognition

5.1 Histograms of Oriented Gradients in 3D

In this section, the HOG3D algorithm and BOW features is described for the action recognition.

5.1.1 HOG3D Features

HOG3D features were developed by Kläser in 2008 [47]. HOG3D features are extracted from within a 3D box centered at a key point to encode both spatial and temporal information. In this algorithm each 3D box is divided into several non-overlapping cells from which the HOG3D features are calculated. The final feature for each box is the concatenation of all cell features in that box. Each cell is further divided into several sub-cells to compute spatial-temporal histogram. The overview of the feature extraction process is shown in Figure 5.1. For each sub-cell, the histogram is obtained by projecting the three mean-gradients $(dx, dy$ and $dt)$ to the icosahedron surfaces, as shown in Equation 5.1 and 5.2,

$$\vec{g} = \begin{pmatrix} \bar{d}_x \\ \bar{d}_y \\ \bar{d}_t \end{pmatrix} \tag{5.1}$$

$$\vec{h} = P\vec{g} \tag{5.2}$$

where P is the projection matrix (10 × 3 for binning to half orientation and 20 × 3 for full orientation). In half orientation (used in this work), only the absolute value of the projected gradient is kept. The projected gradients are then subtracted by a threshold computed in Equation 5.3 and all negative values are set to zero. Then, the histogram vector is normalized in Equation 5.4. Every group of gradient vectors (dx, dy, dt) generates 10 sub-cell histogram values in half orientation and 20 histogram values in full orientation. This thesis uses half orientation throughout all the experiments.

$$threshold = \frac{1.618034}{\sqrt{\sum g_i^2}} \tag{5.3}$$

$$\vec{h}_{norm} = \frac{\sqrt{\sum g_i^2}}{\sum h_j}\vec{h} \tag{5.4}$$

Histogram vectors in a single cell are accumulated from sub-cell histogram using a vector add operation and then normalized again by L2 normalization shown in Equation 5.5.

$$\vec{H}_{norm} = \frac{1}{\sqrt{\sum H_i^2}}\vec{H} \tag{5.5}$$

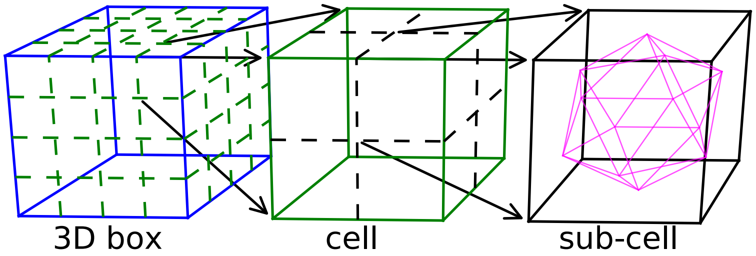

Figure 5.1: Illustration of HOG3D box, cells and sub-cells.

5.1.2 Gradient Computation and Integral Video

As described previously, the histograms are computed by using average gradients in three directions (x, y, t) . The gradients are computed using a simple mask $[-1, 1]$ as in Equation 5.6:

$$\begin{cases} dx = p(x+1, y, t) - p(x, y, t) \\ dy = p(x, y+1, t) - p(x, y, t) \\ dt = p(x, y, t+1) - p(x, y, t) \end{cases} \tag{5.6}$$

Note that edge pixels are replicated (gradients are set to 0 at edges). Integral video is used to rapidly compute average gradients within sub-cells. The integral video is an extension of the popular integral image method proposed by Viola and Jones [93]. Integral video has been shown to be an efficient method to extract spatio-temporal features in previous works [45, 100]. In integral video, the integration value for a gradient at location (x', y', t') is the sum of all gradient values at current and previous locations as shown in Equation 5.7.

$$I_d(x', y', t') = \sum_{t=0}^{t'} \sum_{y=0}^{y'} \sum_{x=0}^{x'} d(x, y, t) \tag{5.7}$$

In HOG3D, the integral videos for each of the three gradients are computed using Equation 5.7. With a given 3D sub-cell at location (x, y, t, w, h, l), the average gradient is computed using Equation 5.8. Note that x, y and t are the smallest column, row and time index in a sub-cell respectively and w, h and l are the width, height and length of the sub-cell.

$$
\begin{aligned}
\bar{d} = [I_d(x-1, y-1, t+l) &+ I_d(x+w, y+h, t+l) - \\
I_d(x+w, y-1, t+l) &- I_d(x-1, y+h, t+l)] - \\
[I_d(x-1, y-1, t-1) &+ I_d(x+w, y+h, t-1) - \\
I_d(x+w, y-1, t-1) &- I_d(x-1, y+h, t-1)]
\end{aligned}
\tag{5.8}
$$

5.1.3 Dense Sampling and Multi-Scale Processing

As mentioned previously, dense sampling algorithm extracts key points at regular locations by moving a 3D box across the video at a constant stride. In this experiment, the stride is 50% of the box size that is any two adjacent boxes have 50% overlap. To further increase the feature diversity (and increase recognition accuracy), features are extracted at multiple scales. Instead of resizing the original frames/images, the 3D box is enlarged by approximately $\sqrt{2}$ times (each side) until the box is larger than the frame size. Thus, a single integral video computation can be used for all spatio-temporal scales. Since dealing with multiple scaled images using shared hardware is difficult [106, 63], this design also simplifies the hardware design that will be discussed later. In all experiments, the box is only resized in spatial dimension at seven different scales with box size of $24, 32, 48, 64, 96, 136, 192$ pixels. The box size is fixed at 16 frames in temporal dimension as multiple temporal scales has shown little impact to the final classification result in [98].

5.1.4 Bag-of-Words Features

BOW feature is a higher level video representation built upon pixel-level features (such as HOG3D). This method was inspired by document classification, where a histogram of "words" (also called vocabularies) are generated to model the document [18]. In this method, the model is built using their occurrence in the document regardless of the order. For computer vision applications, a visual vocabulary is computed by a clustering algorithm (e.g. K-means or KD-Tree) using the extracted features. Each HOG3D feature vector in a video clip is binned into its closest vocabulary to form a BOW feature. The BOW method has been widely used in many of the latest HAR algorithms [53, 51, 100, 47, 96, 79, 97]. In the evaluation process, k-means is chosen as the clustering algorithm and 1,000 vocabularies (i.e 1,000 cluster centers in K-means) are used for all recognition tasks.

5.1.5 Training and Classification

To recognize multiple actions in a video clip, the BOW features are passed to a multi-class SVM classifier for classification. The SVM classifier creates a large margin around the decision boundary (hyperplane) to achieve maximum classification performance [91]. Specifically in a linear SVM classifier, the final confidence score is the dot-product of the trained classification vector (normal vector to the hyperplane) $\vec{W}$ and the BOW feature vector $\vec{V}$ plus a constant intercept term s_0, as shown in Equation 5.9.

$$s = \vec{W} \cdot \vec{V} + s_0 \tag{5.9}$$

The decision boundary can be non-linear if the “kernel trick” is applied to the SVM leading to improved recognition rate [50, 81] but also increased computational complexity. In the evaluation, both linear and χ^2 kernels are used to test the recognition rate as in Equation 5.9 and 5.10 respectively.

$$s = \sum \frac{2w_i \cdot v_i}{w_i + vi} + s_0 \tag{5.10}$$

Originally, SVM classifiers were designed for binary classification. To extend it for multi-class cases, the generally accepted “one versus one” method has been adopted [48, 49]. In this method, one classifier is built for each pair of actions, and the final classification decision is the highest count action after evaluating all classifiers. For example, if there are six actions to recognize, 15 classifiers (C_2^6) are modeled. The outcome of such classifier is a histogram of action counts, and the highest count (maximum count is *five* if all classifier predictions are correct) will be the decision.

Due to limited number of data in all benchmarks for training and testing, cross-validation is used to evaluate the performance of the detector in this work. Cross-validation is a technique commonly used in machine learning to estimate the accuracy of the predictive model. In cross-validation, the entire dataset is divided into two groups, training data and test data. A predictive model is built using the entire training data and then applied against the test data. The average recognition rate is computed by comparing the predicted labels with the ground truth. Leave-one-out cross-validation is a special case of cross-validation that uses one data as test group and the rest as training. The process is repeated until all data are used as test set. In the evaluation, both regular corss-validation and leave-one-out cross-validation are used based on dataset specification.

5.2 Fixed-Point HAR

In this section, the HAR benchmarks used for evaluation of the fixed-point recognition is reviewed. The effect of reduced bit-width on the HAR applications in k-mean clustering, SVM training and classification are also studied.

5.2.1 HAR Evaluation Benchmarks

Three different benchmarks with increasing difficulty are used to evaluate this HAR implementation: KTH [82], UCF11 [60, 61] and UCF50 [79]. KTH benchmark is a collection of videos with six different human actions. This dataset contains 599 video clips with frame size of 160×120 pixels. The videos were taken under controlled environment with good lighting condition and little camera motions. In addition, the dataset contains only one subject (totaling 25 subjects) per video. Thus, the recognition on this dataset is relatively trivial. Example of the six actions are shown in Figure 5.2.

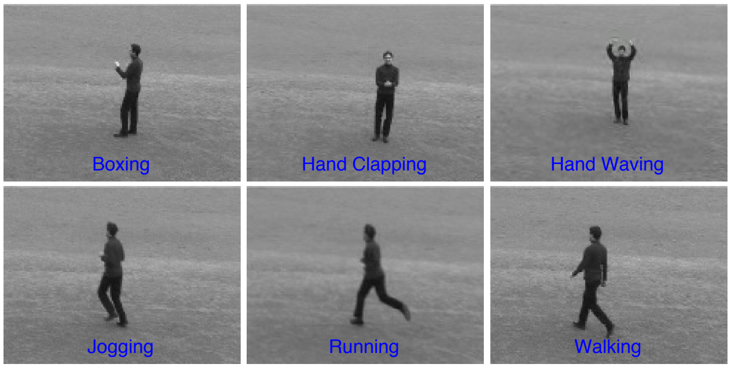

Figure 5.2: Actions in KTH dataset.

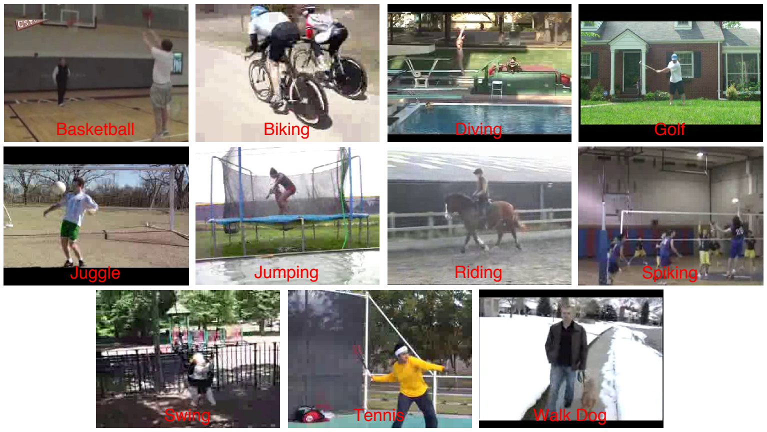

Figure 5.3: Actions in UCF11 dataset.

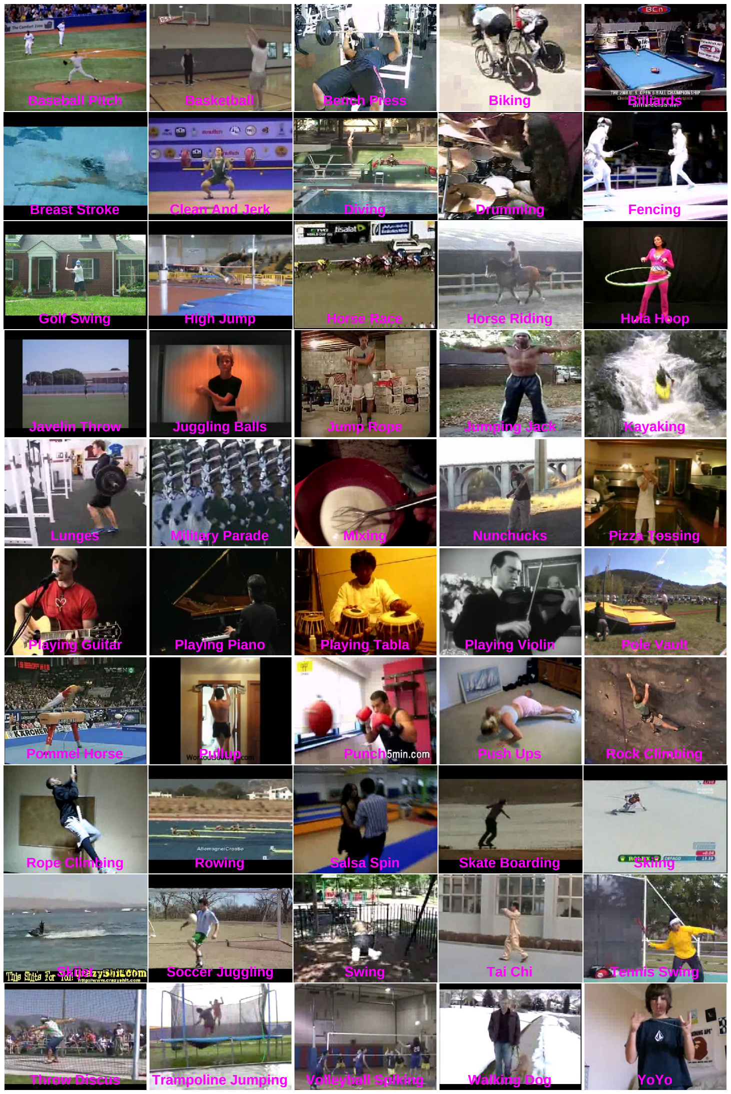

Figure 5.4: Actions in UCF50 dataset.

The UCF11 consists of 1,600 video clips with 11 different actions. Every action is divided into several different groups based on similarity of videos in terms of actors, background, and/or view point. This dataset is very challenging as there is a large number of variations in camera motion, background, subjects, and object scales. The frame size of the video is 320 × 240 pixels. Sample frames of the actions are shown in Figure 5.3. UCF50 is an extension of the UCF11 by adding 39 more actions with a total of 6,680 video clips. This dataset is the most difficult among all three benchmarks due to the large number of actions to be recognized. The sample actions in UCF50 dataset are shown in Figure 5.4.

5.2.2 Fixed-Point Experiments

Both floating-point and fixed-point HOG3D feature extraction are implemented in C++. The floating-point code is based on the original author [47, 46] with a dense-sampling implementation developed in this work. As discussed in Section 5.1, a 3D box is moved at stride of 50% of the box size to extract the features. Moreover, the spatial size is increased approximately $\sqrt{2}$ times until the box is larger than the original frame size. The two spatial directions always have the same size and the box's temporal length is fixed to 16 frames. In this work's dense-sampling algorithm, the smallest spatial size in the experiment is 24 pixels and the box size is always rounded to the nearest integer that is a multiple of sub-cells per dimension. As shown in Figure 5.1, the parameters chosen in this work is four cells per box per direction and two sub-cells per cell per direction, thus the box size should always be a multiple of *eight* pixels. Accordingly, the dimension of the HOG3D feature is 640 with half-orientation ($4 \times 4 \times 4 \times 10$). The number of frames in each video clip is fixed to 396 frames for KTH, 96 frames for UCF11 and 80 frames for UCF50.

The floating-point code are implemented in double-, single- and half-precision floating point to test the outcome at different data precision. The fixed-point feature extraction is similar to the floating-point version but with all computations performed in fixed-point. In the fixed-point evaluation program, the bit-width can be passed as an input argument at run-time for processing in different precision. The data size was obtained by sampling the three benchmarks at every step of the computation. The data flow of the activity recognition procedure and the bit-width at different steps are shown in Figure 5.5, where n is the bit-width (excluding sign bit, if applicable). Negative fractional values for small n in Figure 5.5 is automatically set to 0 in the implementation. In the experiment, a large range of bit-width from 27-bit down to 6-bit are used. The upper bound is chosen to make sure no intermediate fixed-point data exceeds 64-bit during any stage of the computation. Due to vast amount of features extracted from each dataset, features used for K-means clustering are randomly selected. For KTH and UCF11, 400 and 200 features are selected from a video clip respectively. UCF50 is significantly larger than the others, thus, 10,000 features per action category are randomly sampled for clustering. Fixed-point and half-precision HOG3D features are converted to single-precision floating point for k-means clustering. All clustering are set to terminate after reaching one million iterations. For fixed-point data, the centers are converted back to their respective bit-width.

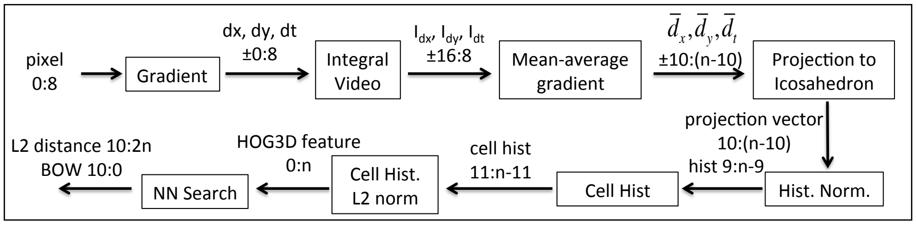

Figure 5.5: HOG3D data-flow diagram and key parameter data sizes (integer:fractional) used in the implementation.

BOW features are built using brute-force nearest neighbor search method. The nearest cluster center for a feature vector is determined by comparing the L2 distance with all the 1,000 centers. What's more, the nearest neighbor search is also implemented in double- and single-, precision floating-point as well as fixed-point. Half-precision features are converted to single-precision for nearest neighbor search.

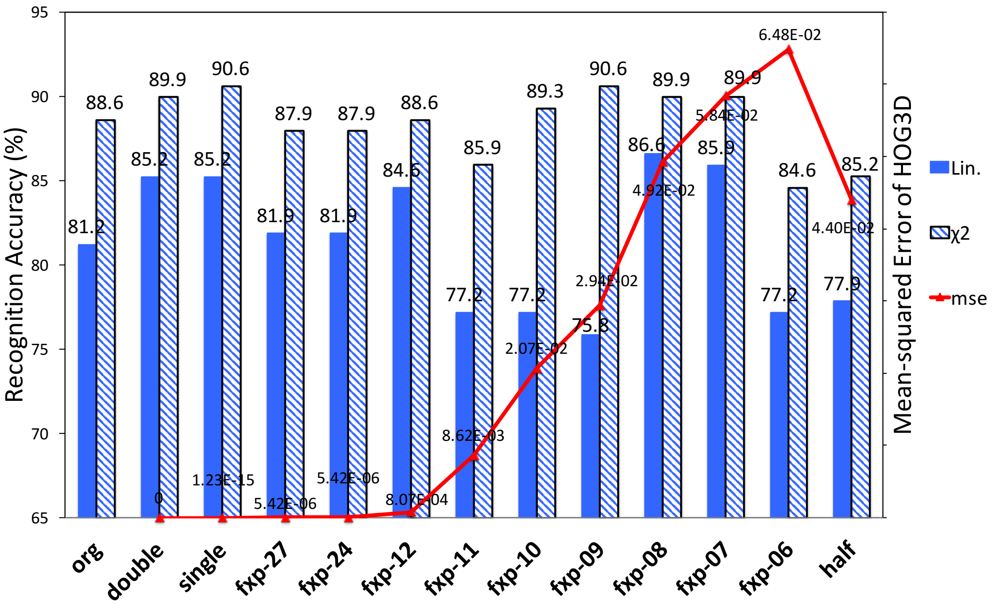

Figure 5.6: KTH dataset recognition results in linear and χ^2 kernel and MSE.

To evaluate the accuracy of the activity recognition, this work has followed the original experimental settings from the authors of the benchmarks. A modified version of LIBSVM (added χ^2 kernel) is used for classifier training [13]. This library uses double-precision floating-point for all training. For KTH dataset, nine subjects (2,3,5,6,7,8,9,10, and 22) are used as test group and the rest as training group. For both UCF11 and UCF50 datasets, leave-one-group-out cross-validation is used (since videos at the same group are similar, they are used in training/testing together). Note that because the UCF50 is a superset of UCF11, only the 39 actions that are not included in UCF11 dataset are evaluated.

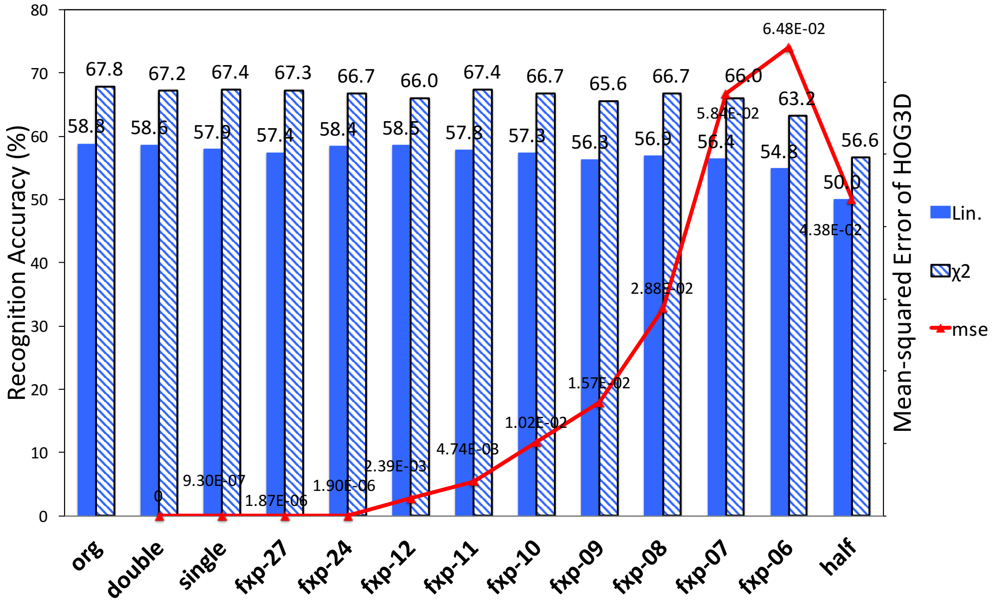

Figure 5.7: UCF11 dataset recognition results in linear and χ^2 kernel and MSE.

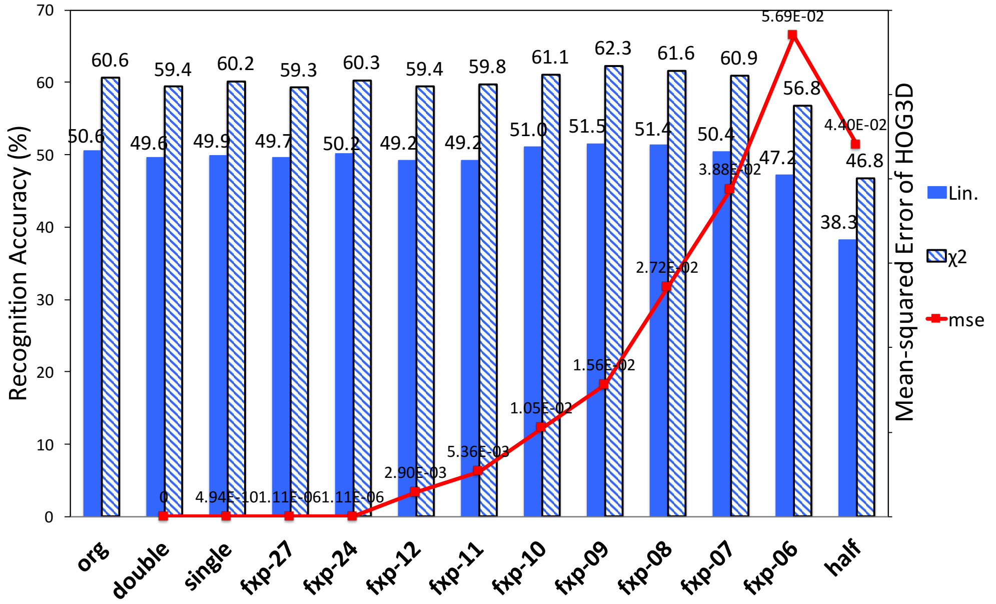

Figure 5.8: UCF50 dataset recognition results in linear and χ^2 kernel and MSE.

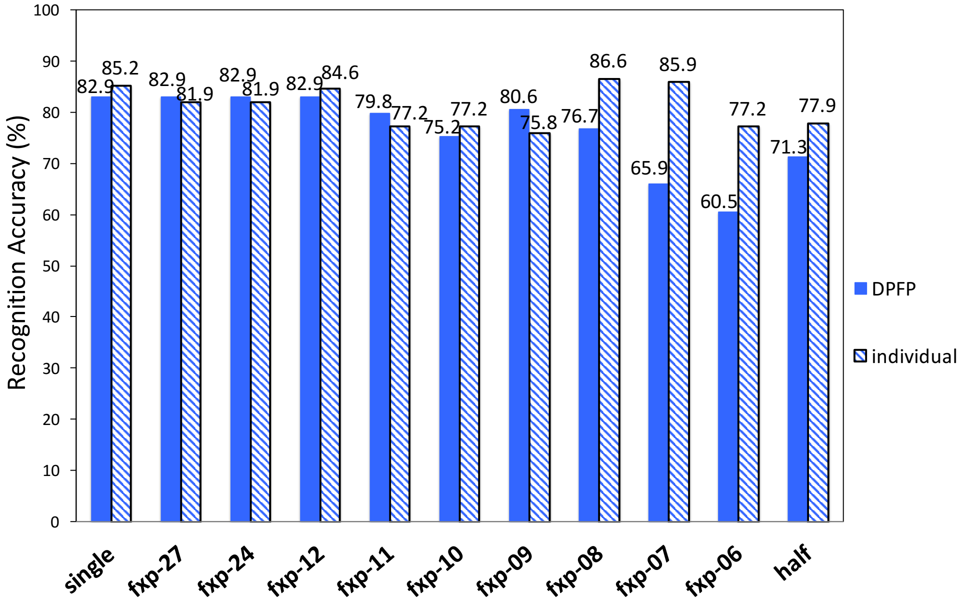

Figure 5.9: KTH dataset recognition results using DBFP and individually trained centers.

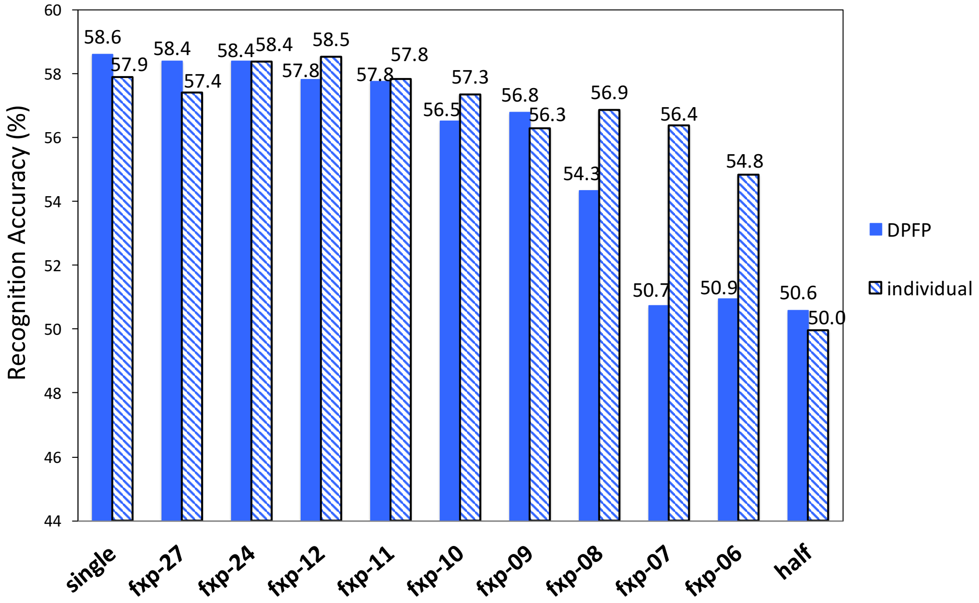

Figure 5.10: UCF11 dataset recognition results using DBFP and individually trained centers.

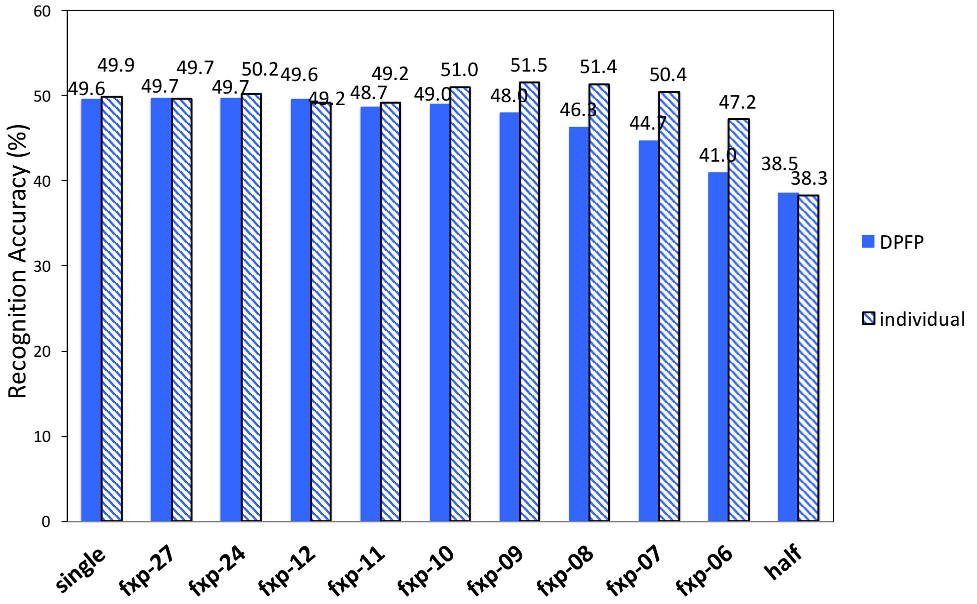

Figure 5.11: UCF50 dataset recognition results using DBFP and individually trained centers.

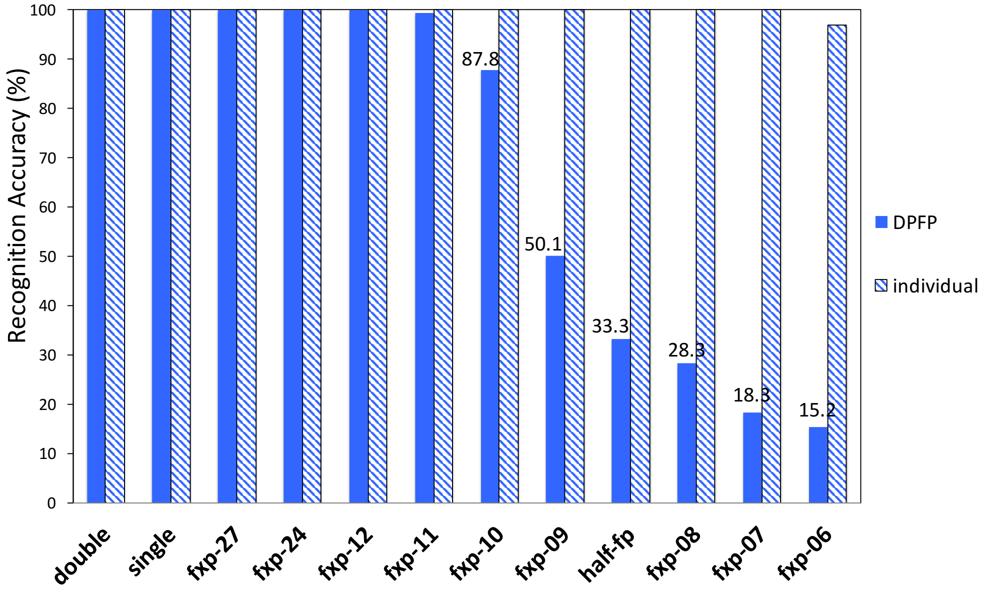

Figure 5.12: KTH dataset recognition results using DPFP SVM model and individually trained SVM model.

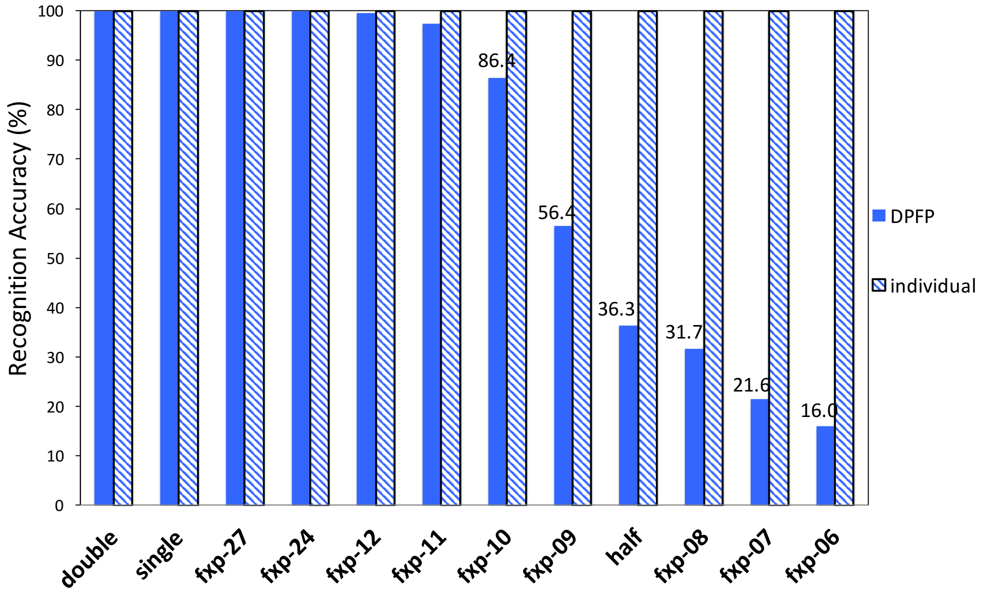

Figure 5.13: UCF11 dataset recognition results using DPFP SVM model and individually trained SVM model.

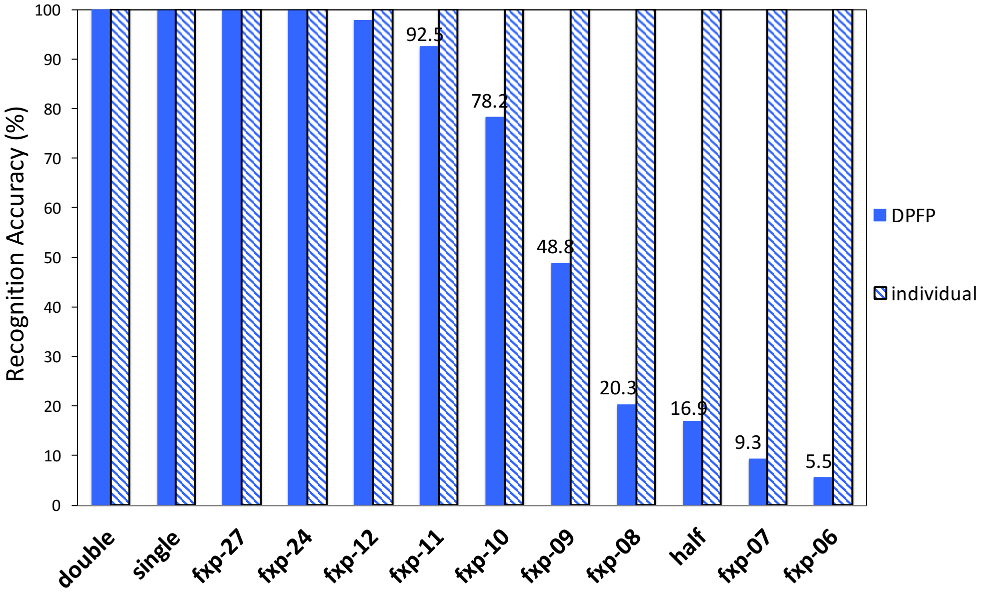

Figure 5.14: UCF50 dataset recognition results DPFP SVM model and individually trained SVM model.

5.2.3 Recognition Results

The recognition accuracy are assessed by using all processing steps in fixed-point (fxp-) and compared the results with float-point data, as shown in Figures 5.6, 5.7, and 5.8. Both linear and χ^2 kernel SVM classification results are reported in this thesis. The “org” evaluation uses the original authors’ code and configurations in feature extraction that uses floating-point indexed and sized 3D boxes to sample the video. In addition, the mean-squared error (MSE) of the HOG3D features is calculated by using the double-precision floating-point (DPFP) feature as ground truth. As shown in Figure 5.6, the accuracy at different bit-width fluctuates a lot. However, for UCF11 and UCF50, the lower bit-width is very stable as shown in Figures 5.7 and 5.8. The fluctuation in KTH dataset is likely due to the limited number of training/testing samples. For UCF50 dataset, the fxp-8 slightly outperforms the DPFP recognition even though the MSE in feature extraction increased by several orders of magnitude. The half-precision recognition performs worst in all cases. The MSE at fxp-8 ranges from 2.7% to 4.9% and increases by about 13 orders of magnitude from the MSE of single-precision float, while the recognition accuracy remains relatively the same. Additionally, the MSE for half-precision float is between fxp-8 and fxp-7. However, half-precision recognition accuracy is well below the lower bit-width fixed-point recognition for all test cases. This is due to the error propagation in the integral video stage. On one hand, fixed-point feature extraction does not suffer precision loss during integral video as the data range is carefully selected to avoid overflow. On the other hand, the integral video using half-precision may suffer large amounts of information loss due to the limited range and precision of half-float. This information loss is further amplified at later stages.

To study how the recognition performs well under low bit-width, further experiments are performed to investigate the effect of Kmeans-clustering. In addition to performing recognition by finding clustering centers in each bit-width ("individual" in Figures 5.9, 5.10, and 5.11), the effect of nearest neighbor search is investigated by using clustering centers trained from DPFP features ("DPFP" in Figures 5.9, 5.10, and 5.11). All other evaluation parameters remains the same as in previous evaluations. Only χ^2 kernel SVM result in cross-validation is used for comparison. As shown in Figures 5.9, 5.10, and 5.11, recognition accuracy drops significantly as the bit-width decreases when using clustering centers from DPFP features. Therefore, rebuilding clustering centers for individual bit-width has an important contribution to overall recognition accuracy at low bit-width.

What's more, the effect of SVM training is evaluated. Similar to the K-means study, an SVM prediction module (with χ^2 kernel) is built using DPFP features in HOG3D and K-means. Then the same model ("DPFP" in Figures 5.12, 5.13, and 5.14) is applied to the fixed-point features extracted from HOG3D plus BOW features using DPFP clusters. Additionally, individual SVM model is trained using the fixed-point plus BOW features for comparison. Note that since different SVM classifiers are compared, no cross-validation is used. All features are applied to both training and testing. When the same data is applied to both training and testing, the recognition accuracy will normally be 100%. Figures in 5.12, 5.13, and 5.14 show the comparison result. The dramatic accuracy difference between individually trained SVM models and DPFP SVM model shows that when retraining SVM using reduced precision features, the internal prediction model has been changed. This model change is significant enough, under low bit-width (fxp-10 and below), to cause the recognition failure even with the same training data.

Based on above analysis, the good recognition accuracy under low bit-width is attributed to three main factors: (1) Small information loss at early stage of feature extraction. (2) Performing K-means clustering for the reduced bit-width to build centers better suited for that bit-width. (3) Re-train the SVM classifier at the end to generate classification models specific for the data. This findings are not limited to FPGA-based applications but are relevant all hardware resource constrained embedded or real-time processing learning systems to achieve faster and more power-efficient computation.

5.3 FPGA Implementation

This section provides detailed FPGA implementation. The FPGA-based feature extraction module is implemented on Convey HC-2ex machine [2]. The system is composed of two Intel Xeon E5-2643 CPUs and four Xilinx Virtex-6 LX760 FPGAs. Both CPUs and FPGAs share a globally addressable 192-GB virtual memory, 64-GB on FPGA side and 128-GB on CPU side. FPGA memory is connected to CPUs via PCIe 3.0 x16. Each FPGA has 16 64-bit memory channels at 150 MHz controlled by eight memory controllers. The memory subsystem provides a highly parallel and high bandwidth (19.2 GB/s per FPGA) connection between FPGAs and physical memory. 8-bit fixed-point is used to perform HOG3D feature extraction and nearest neighbor search. The implementation is a complete end-to-end solution that reads raw pixels in gray-scale and generates BOW features as output. Each video has 97 320×240-pixel frames. The BOW feature is a histogram consisting 1,000 bins. 10 memory channels are used in the system, two for input and eight for output. The entire implementation is fully pipelined so that all modules can immediately start processing the next group of data after the first is streamed in.

5.3.1 Integral Video and Gradient Vector

As the first step of HOG3D feature extraction, gradients are computed from pixel values using Equation 5.6. One memory channel is used for raw pixel input. Because temporal gradients dt needs two frames to compute the final gradient, to avoid using on-chip memory to store the entire previous frame, two frames are accessed at the same time. The input controller generates addresses for "current" frame and "next" frame alternatively. "Next" frame is only used in computation of dt while "current" frame is used in all gradients. Memory resource is minimized as only one line buffer is used in computing dy. Three gradients are then sent to integral video module for further processing.

As described in Section 5.1, integral video is computed by summing up all gradient values in a video to sppedup the calculation of average gradients. The integration value range is determined by both the size of the frame and the number of frames. When the frame number or frame size increases, the data range will grow accordingly. This is especially a problem for FPGA-based fixed-point implementation where small data range is more desirable. Observe that Equation 5.8 can be re-written as $\bar{d} = J(t+l) - J(t)$, where $J(t) = I_d(x, y, t) + I_d(x+w, y+h, t) - I_d(x, y+h, t) - I_d(x + w, y, t)$. Also based on the definition of integral video, $J(t + l) - J(t)$ is the sum of all pixels between t and t+l (excluding t) and in the area of rectangle $(x, y, w, h\ )$. In the configuration, l is always two and t are constant indices (0, 2, 4, 6, ...). Note that in integral video/image, index 0 is a frame/row/column with all pixels set to 0, not the first frame/row/column. Hence, the gradient vector $\bar{d}$ can be computed using the new integral video INT_d which is the sum of two consecutive frames as shown in Equation 5.11 and 5.12.

$$\begin{aligned}\bar{d} = & INT_d(x, y) & + INT_d(x+w, y+h) - \\ & INT_d(x+w, y) & - INT_d(x, y+h)\end{aligned} \tag{5.11}$$

$$INT_d(x', y') = \sum_{t=t_1}^{t_2} \sum_{y=0}^{y'} \sum_{x=0}^{x'} d(x, y, t) \tag{5.12}$$

Moreover, for each scale, a fixed stride is used to compute gradient vectors from integral video. For example, for the first scale (scale 0), pixels in every third row/column are accessed and for the second scale (scale 1) pixels in every fourth row/column are used in computation as shown in Figure 5.15. To maximize data sharing and reduce on-chip buffering, integral video is computed for a group of scales. Within a group, all scales share some of the pixels. For example, scale $1, 3$ all access pixels that are multiply of four (stride of four). All groups share a single line-buffer that stores all integration values at previous row. However, each group uses one FIFO to store previous frame's integration values at those locations to reduce the number of saved pixels. As a result, the video integration module does not need to buffer every pixel in the previous frame. In the configuration with seven scales (stride $3, 4, 6, 8, 12, 17, 24$ respectively), only 12761 out of 76800 (320×240) pixels in a frame are saved in the FIFO for previous frame.

The architecture of the integral video and gradient vector computation is shown in Figure 5.15. Integral image is first computed for each frame of the pixel gradients (by using a line-buffer). Within each line, pixels are sent to frame FIFO if they belong to that scale group (multiply of 3 for scale 0, 2, 4, 6, multiply of 4 for scale 1, 3 and multiply of 17 for scale 5). Also note that as described previously, only odd indexed frames are sent to the FIFO to be combined with even indexed frames for two-fame video integration. After obtaining the integral video, values are checked if they belong

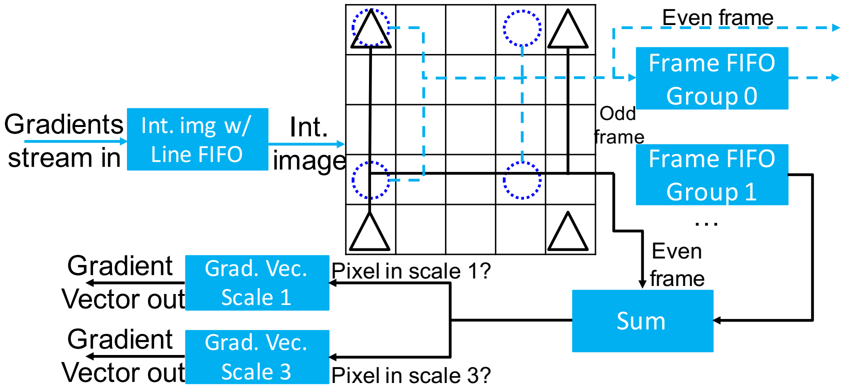

Figure 5.15: Hardware architecture to compute two-frame integral video and gradient vectors (scale 1, 3 are shown).

to a specific scale to compute gradient vector(e.g. scale 3 has a stride of 6, it will not use pixels at index 3).

The pixel gradient, video integration and gradient vector computation module is implemented in C++ and synthesized by Xilinx Vivado HLS. The input address generator unit is directly implemented in Verilog VHDL.

5.3.2 HOG3D Feature Extraction

To better optimize the fixed-point arithmetic operations (in Equation 4.4 and 5.2), gradient projection is implemented in Verilog HDL. The procedure of generating sub-cell HOG3D features are shown in Figure 5.16 ($\vec{h}_{norm}$ in Equation 5.4). The $L2 - norm$ of the gradient vectors ($\sqrt{\sum g_i^2}$ in Equation 5.3 and 5.4 is computed in parallel with the projection. The norm coefficient is to compute $\frac{\sqrt{\sum g_i^2}}{\sum h_j}$ in Equation 5.4. Note that the entire experiment uses half orientation where each HOG3d feature consists of 10 elements. Because $L2 - norm$ of gradient vectors have a very large range, all values are converted to half-precision floating-point before performing division in

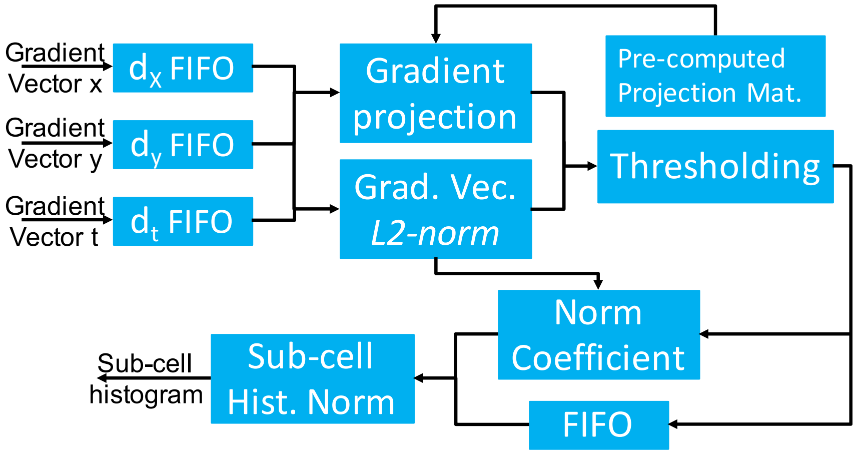

Figure 5.16: Block diagram of gradient projection module on FPGA.

computing normalization coefficient and then converted back to fixed-point (division by 0 will set the result coefficient to 0). To expedite processing, seven gradient projection units are instantiated (one for a scale).

Sub-cell HOG3D features are then accumulated into a single cell by a simple vector add. In the experiment configuration, each cell consists of $2 \times 2 \times 2$ sub-cells, as shown in Figure 5.1. There is no overlapping between adjacent cells in any dimension. Same as the gradient projection, cell histogram is also generated per scale. Three FIFOs are used in each cell histogram module: previous column, previous row, and previous frame as shown in Figure 5.17. Like pixels in the image, the sub-cell histogram is also arranged in columns, rows and frames. Each column contains one sub-cell features (10 elements) and each row contains all sub-cells extracted in a row of sliding windows from two-frame integral video. Similarly, a frame of sub-cell is all the features from a two-frame integral video. The accumulation of sub-cell histogram is equivalent to a sliding box of

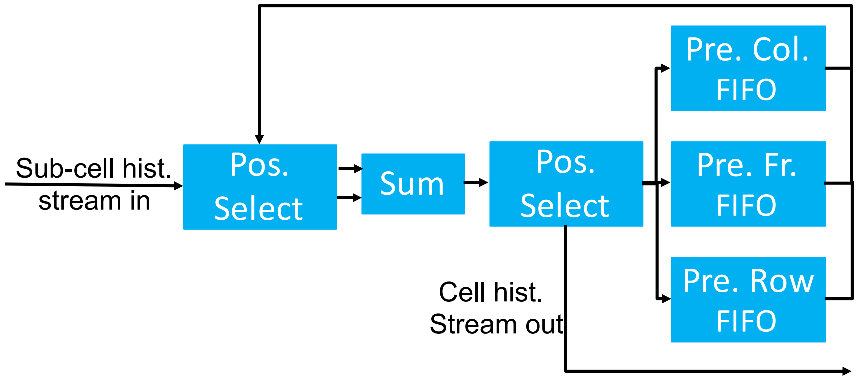

Figure 5.17: Block diagram of cell histogram generation module.

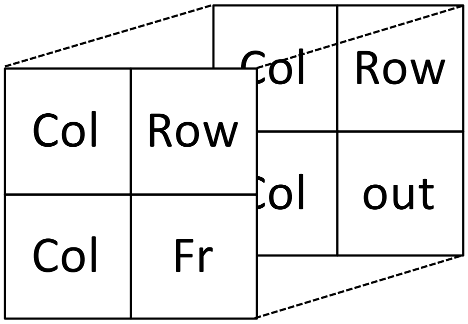

Figure 5.18: Diagram of destination FIFOs for each location in a cell. Out means the cell histogram is streamed out.

size $2 \times 2 \times 2$ without any overlapping. The accumulation module determines the location of current sub-cell histogram inside a sliding box (see cell diagram in Figure 5.17. Based on the histogram location, it reads one of the three FIFOs and adds current feature to the values in the FIFO. Then, it sends the accumulated to appropriate destination FIFO as shown in Figure 5.18. After accumulating histograms in all eight locations, cell histograms are streamed out. This module is implemented in C++ and synthesized by Xilinx Vivado HLS.

Cell histograms are normalized according to Equation 5.5. The normalization unit is implemented in Verilog HDL. The normalization module is implemented using square root, division, and multiplication modules. Similar to the $L2 - norm$ in Figure 5.16, the histogram squares are summed, then sent to square root module. Finally, the reciprocal is computed by the divider core. The normalized histogram value is the multiplication of histogram and the reciprocal value. The entire computation is in performed fixed-point as discussed in Section 5.2.

Cell histograms are directly sent back to main memory on FPGA for constructing HOG3D features and nearest neighbor search. Different scales of histograms are stored linearly one after another. The output controller design is straightforward that generates sequential addresses plus a constant scale offset. Output memory controller is implemented in Verilog HDL. Seven memory channels are used to send HOG3D features into main memory. Note that the entire HOG3D feature extraction is computation-bounded. The input and output memory channels are sufficient to deliver/send data.

5.3.3 Nearest Neighbor Search

The input controller of the nearest neighbor search is responsible to generate feature addresses that construct actual HOG3D features. The HOG3D features are obtained by concatenating

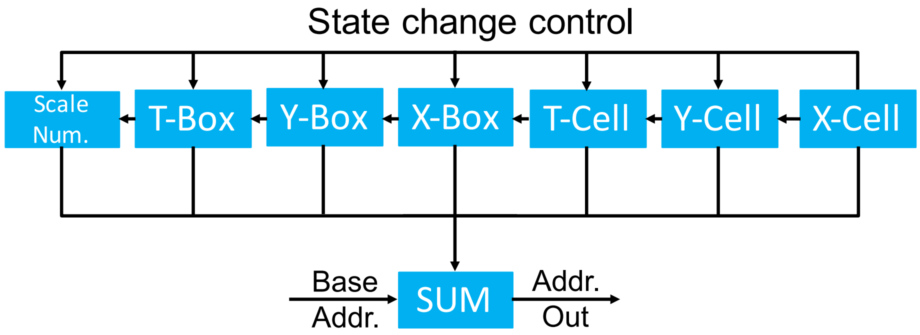

Figure 5.19: Nested state machines control the address offsets generation to construct HOG3D features.

all cell histograms in a 3D box as shown in Figure 5.1. The concatenation is in row-major order and the adjacent boxes have 50% overlapping. The address generation module is analogous to [63]. Different scales are processed sequentially. Seven nested state-machines are used to generate the address offsets for current scale, t-box, y-box, x-box, t-cell, y-cell, and x-cell, as shown in Figure 5.19. "Scale Num." is the outmost state machine while "X-cell" is the the innermost state machine. "X-cell" state machine generates five offsets for 40 consecutive histograms (4 × 10 cell histogram, 8-bit each as shown in Figure 5.5). When each inner state machine reached the end, it notifies all outer state machines so that they can check if a state change is needed. After all state machines reached the end, addresses for all scales are generated. It takes 80 clock cycles to generate all addresses for one HOG3D feature (640 elements), and there are 10241 features per 97-frame video.

The nearest neighbor search module finds the smallest distance between each feature and all 1000 centers discussed in Section 5.2. In HOG3D, each cell contains $4 \times 4 \times 4$ sub-cells. Thus, the HOG3D feature has 640 dimensions (10 features per sub-cell). To find the nearest neighbor, the distance between each feature and 1000 centers are computed and then compared. To maximize the throughput, 640 multipliers are instantiated to compute one distance per clock cycle. HOG3D

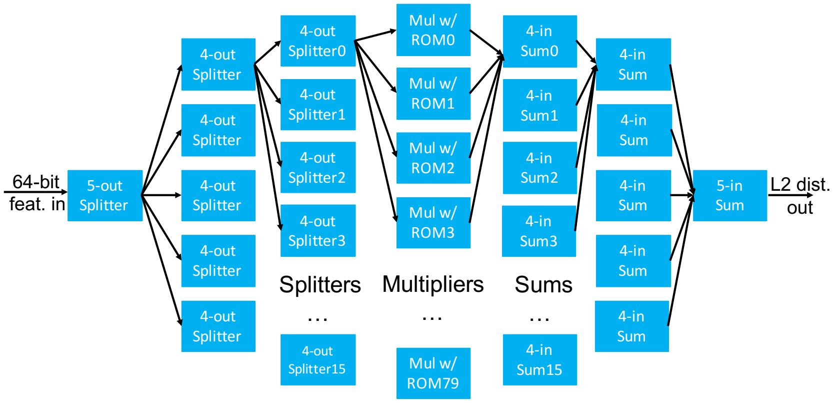

Figure 5.20: Illustration of parallel nearest neighbor Search architecture.

features are loaded from memory and all center data is stored on chip in ROMs. These 640 multiplications are divided into 80 multiplication cores as each memory access will return 8 features as shown in Figure 5.20. The centers are also divided into 80 ROMs (1000 × 64-bit size) based on their position in a feature. Input feature is passed to a three-stage data splitter that sends the feature into appropriate multiplier core. Three stage design reduces number of fan outs in a single stage to better meet timing. Each multiplication core computes computes eight euclidean distances at once and outputs their sum. Another three-stage summation is used to produce the final $L2$ distance between the feature with one of the centers.

The BOW feature is a histogram of 1000 bins. When a minimum distance center of a feature is found, the bin count corresponding to the center will be incremented by one. A streaming histogram accumulation unit is built to process 1000 histogram bins. The diagram of this unit is shown in Figure 5.21. Histogram bins are organized in a chain structure. The center index (bin number) is streamed in from the first bin. It will check if the index is the same as its defined bin value. If the input equals current bin number, its counter is incremented. If it does not equal current

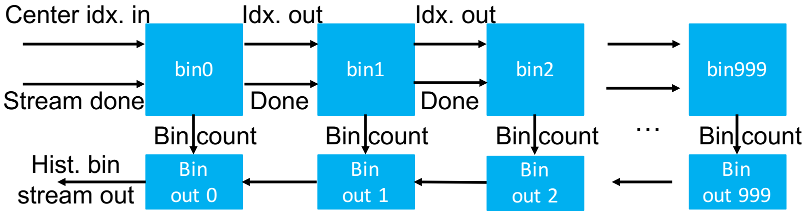

Figure 5.21: Illustration of streaming histogram accumulation unit.

bin, the index will be sent to the output which is the input of next bin. Immediately after all 10241 indices are streamed in, a done signal is streamed from bin0 to bin999 causing accumulated bin values to be streamed out. Once received the done signal, each bin is ready to start accumulating next histogram. Similar to HOG3D feature extraction, nearest neighbor search is computation-bounded.

5.4 Results and Evaluation

In this section the speedup of the feature extraction is evaluated for FPGA and GPU implementation over a highly optimized CPU code. All tests are based on UCF50 dataset with 97 frames per video.

5.4.1 CPU Results

Our CPU implementation uses single-precision floating-point for HOG3D feature extraction and nearest neighbor search. The CPU platform is an Ubuntu machine with two Intel Xeon-E5520 quad-core CPUs and 24-GB RAM. The CPU program is implemented in C++ with Intel TBB library for multi-threading capability. SSE is also enabled in nearest neighbor search. The processing time measured for CPU implementation is the actual feature extraction time. Loading the video from hard drive and writing features to file are not included. The experiment shows that it takes about 4.80 seconds for a multi-core machine to process one video (97 frames).

5.4.2 GPU Results

In this section[1] the single-precision floating point GPU implementation of the HOG3D feature extraction is briefly described. The implementation is running on an Ubuntu workstation with a quad-core Intel i7-860 CPU, 8-GB RAM, and an Nvidia Tesla K20c GPU[4]. The code is compiled with the NVIDIA CUDA toolkit 6.0 with the Basic Linear Algebra Subroutines (cuBLAS) V2.0. The GPU error correcting code capabilities ECC is disabled in all experiments. In this implementation, all major tasks are executed in the GPU. The time measurements do not include the transfer of the videos to the GPU nor the time to allocate space in the GPU global memory. The GPU implementation can process 1,616 frames per second (fps) i.e. 16 videos in 0.96 seconds.

5.4.3 FPGA Results

The entire implementation (including the Convey memory interface) has been synthesized using Xilinx ISE 14.7. All Vivado HLS generated modules are connected to the hand-written Verilog code Using FIFOs. For arithmetic operations, portions of the nearest-neighbor search distance computation (multiplication) are placed into DSPs (26% of the multiplications in nearest-neighbor search) and all other operations in pure logic. Table 5.1 summarizes the synthesis result. In addition to the Virtex-6 FPGA on the Convey HC-2ex machine, the implemented algorithm is also synthe-

[1] The entire details of the GPU implementation are beyond the scope of this thesis. It should be noted however that it is the first such implementation of HOG3D on GPUs.

Table 5.1: FPGA implementation resource utilization.

Attributes	Virtex-6 LX760 (Convey)	Kintex-7 XCU060
Registers	312085 (32%)	214820 (32%)
LUTRam	39987 (30%)	15068 (10%)
LUTs	197025 (41%)	123708 (37%)
36KBram	247 (34%)	265 (25%)
DSPs	168 (19%)	320 (12%)

sized on a Xilinx Kintex-7 Ultrascale FPGA (XCU060) for comparison. The clock on the Kintex-7 FPGA is also set to 150 MHz.

As described in Section 5.3, both HOG3D feature extraction and nearest neighbor search are computation-bounded and memory bandwidth is not fully utilized. Then, the processing speed is determined by the actual number of clock cycles to process data on FPGA. HOG3D feature extraction takes $96 \times 320 \times 240$ clock cycles to process (frame 97 is only used for dt computation, and is processed in parallel with frame 96) which is equivalent to 0.049 seconds at 150MHz frequency. Additionally it takes about 0.068 seconds to process a video in nearest neighbor search (1000 × 10241 clock cycles). Consequently, the overall speed of the FPGA implementation is about 1420.8 frames per second. The summary of all platforms in shown in Table 5.2.

Note that an FPGA implementation of the this application using floating-point data would not only be too large for both FPGAs, its parallelism would also be constrained by the memory bandwidth.

Table 5.2: Human action recognition feature extraction throughput comparison.

Platform	Throughput (fps)	Speedup
CPU	20.2	1
GPU	1,616.0	80
one FPGA-fxp8	1420.8	70
four FPGA-fxp8	5682.8	280

5.4.4 Speed, Power, and Accuracy Trade-off

Our work focuses on the action recognition in camera networks where battery powered cameras are distributed across multiple locations. Information captured by cameras are sent via Wi-Fi. A centralized processing of such information is limited by the available bandwidth, security concerns and the difficulty in processing massively large amounts of data. Thus this thesis proposes to use FPGAs to process raw pixels behind the camera and only send extracted action features back to center servers for classification. In this application, both processing speed and power consumption are critical.

Recently, CNNs have been shown to provide exceptional accuracy on large-scale recognition problems [44, 85, 32, 70]. However, applying a deep neural network for real-time embedded applications remains challenging [25, 105].

The computational complexity of CNNs is significantly higher than that of hand-crafted feature extraction (such as HOG3D). The throughput of the open-source CNN-based HAR algorithm [24] is compared with the implementation of HOG3D on GPU: HOG3D feature extraction is

75X faster on the same benchmarks. Accordingly, using a hand crafted feature (e.g. HOG3D) for real-time and embedded system implementation is at present the better option.

Chapter 6

Conclusion

This thesis focuses on the use optimization of arithmetic computations in computer vision algorithms targeted for real-time and embedded applications and their implementation on FPGAs. To leverage computation parallelism on FPGAs, fixed-point arithmetic is used for all implementations. The benefit of floating-point values is their large data range while fixed-point has only limited range/precision. For embedded applications where hardware sources are limited, using less bit-width in computation is more desirable for higher parallelism and lower energy consumption.

Several computer vision algorithms are systematically evaluated for their performance under reduced-precision fixed-point computation. As a first step, Viola-Jones face detection algorithm is assessed for its performance under reduced data precision: whereas the reference OpenCV [5] code uses double precision floating-point values, by using only five decimal digit (17 bits) fixed-point representation, the detection can achieve the same rates of false positives and false negatives as the reference OpenCV code. By reducing the necessary precision by a factor of 3X to 4X, the size of the circuit on FPGA is reduced by a factor of 12X; hence increasing the number of feature classifiers that can be fit on a single FPGA. This finding leads to a hybrid CPU-FPGA implementation to reduce CPU work-load.

As a second step, this work evaluates the HOG object detection algorithm using the *full-image evaluation methodology* to explore the FPGA implementation of HOG under reduced bit-width. This approach lessens the required area resources on the FPGA and increases the clock frequency and hence the throughput per device through increased parallelism. The study finds that by reducing the bit-width to some extend, the detection precision will be increased while the recall is decreased and leads to a similar detection result as the original floating-point implementation. The single FPGA implementation in this work achieves a 68.7x higher throughput than a high-end CPU, 5.1x higher than a high-end GPU, and 7.8x higher than the same implementation using floating-point on the same FPGA. A power consumption comparison for different platforms shows our fixed-point FPGA implementation uses 130x less power than CPU, and 31x less energy than GPU to process one image.

In addition to object detection algorithms, this thesis also investigates the acceleration of action recognition, specifically a human action recognition (HAR) algorithm. In the evaluation process, one step further is taken to train the classifier with reduced-precision data. Experiment results show that this re-training process can "compensate" the precision loss in feature extraction and lead to the usage of lower bit-width in hardware implementation. Based on this result, an FPGA-based HAR feature extraction is implemented for near camera processing using 8-bit fixed-point data. This implementation, using a single Xilinx Virtex 6 FPGA, achieves about 70x speedup over multicore CPU and is only about 12.5% time slower than a highly optimized GPU implementation.

The fixed-point assessment of computer vision algorithms presented in this thesis is based on hand-crafted feature extractions. As the deep-learning method becomes popular, it is necessary to explore the feasibility of using reduced precision data in neural networks for better detection/recognition accuracy in future work. For embedded applications where processing speed is important, it is desirable to use a hand-designed and neural network hybrid model as discussed in [32].

Bibliography

[1] An annotated (tracked) sequence of a talking face. http://www-prima.inrialpes.fr/FGnet/data/01-TalkingFace/talking_face.html. Accessed: 2016-02-02.

[2] Convey Computers. www.conveycomputers.com. Accessed: 2016-01-17.

[3] Face and gesture recognition working group. http://www-prima.inrialpes.fr/FGnet/. Accessed: 2016-02-02.

[4] NVIDIA's next generation CUDA compute architecture: Kepler GK110. http://www.nvidia.com. Accessed: 2016-01-17.

[5] Open source computer vision library (OpenCV). http://opencv.org/. Accessed: 2016-02-02.

[6] YouTube now gets over 400 hours of content uploaded every minute. http://www.tubefilter.com/2015/07/26/youtube-400-hours-content-every-minute/. Accessed: 2015-05-26.

[7] A Ahilan and E A K James. Design and implementation of real time car theft detection in FPGA. In *Advanced Computing (ICoAC), 2011 Third International Conference on*, pages 353–358. IEEE, December 2011.

[8] Jason D Bakos. High-Performance heterogeneous computing with the convey HC-1. *Comput. Sci. Eng.*, 12(6):80–87, 1 November 2010.

[9] S Bauer, S Kohler, K Doll, and U Brunsmann. FPGA-GPU architecture for kernel SVM pedestrian detection. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 61–68. IEEE, June 2010.

[10] A Benedetti and P Perona. Bit-width optimization for configurable DSP's by multi-interval analysis. In *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on*, volume 1, pages 355–359 vol.1. IEEE, October 2000.

[11] B Bilgic, B K P Horn, and I Masaki. Fast human detection with cascaded ensembles on the GPU. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 325–332. IEEE, June 2010.

[12] C Blair, N M Robertson, and D Hume. Characterizing a heterogeneous system for person detection in video using histograms of oriented gradients: Power versus speed versus accuracy. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 3(2):236–247, June 2013.

[13] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[14] Chuan Cheng and C Bouganis. An FPGA-based object detector with dynamic workload balancing. In *Field-Programmable Technology (FPT), 2011 International Conference on*, pages 1–4. IEEE, December 2011.

[15] Junguk Cho, B Benson, S Mirzaei, and R Kastner. Parallelized architecture of multiple classifiers for face detection. In *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, pages 75–82. IEEE, July 2009.

[16] O G Cula and K J Dana. Compact representation of bidirectional texture functions. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–1041–I–1047 vol.1. IEEE, 2001.

[17] N Dalal and B Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893 vol. 1. IEEE, June 2005.

[18] Scott C Deerwester, Susan T Dumais, Thomas K Landauer, George W Furnas, and Richard A Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.

[19] P Dollar, V Rabaud, G Cottrell, and S Belongie. Behavior recognition via sparse spatio-temporal features. In *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*, pages 65–72. IEEE, October 2005.

[20] P Dollar, C Wojek, B Schiele, and P Perona. Pedestrian detection: A benchmark. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 304–311. IEEE, June 2009.

[21] P Dollar, C Wojek, B Schiele, and P Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4):743–761, April 2012.

[22] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(8):1532–1545, August 2014.

[23] Piotr Dollár, Serge Belongie, and Pietro Perona. The fastest pedestrian detector in the west. In *BMVC*, volume 2, page 7. Citeseer, 2010.

[24] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. 17 November 2014.

[25] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. ShiDianNao: Shifting vision processing closer to the sensor. *SIGARCH Comput. Archit. News*, 43(3):92–104, June 2015.

[26] Markus Enzweiler and Dariu M Gavrila. Monocular pedestrian detection: survey and experiments. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(12):2179–2195, December 2009.

[27] A Ess, B Leibe, K Schindler, and L Van Gool. A mobile vision system for robust multi-person tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, June 2008.

[28] Mark Everingham, Andrew Zisserman, Christopher K I Williams, Luc Van Gool, Moray Allan, Christopher M Bishop, Olivier Chapelle, Navneet Dalal, Thomas Deselaers, Gyuri Dorkó, Stefan Duffner, Jan Eichhorn, Jason D R Farquhar, Mario Fritz, Christophe Garcia, Tom Griffiths, Frederic Jurie, Daniel Keysers, Markus Koskela, Jorma Laaksonen, Diane Larlus, Bastian Leibe, Hongying Meng, Hermann Ney, Bernt Schiele, Cordelia Schmid, Edgar Seemann, John Shawe-Taylor, Amos Storkey, Sandor Szedmak, Bill Triggs, Ilkay Ulusoy, Ville Viitaniemi, and Jianguo Zhang. The 2005 PASCAL visual object classes challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, Lecture Notes in Computer Science, pages 117–176. Springer Berlin Heidelberg, 2006.

[29] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Ann. Stat.*, 28(2):337–407, April 2000.

[30] Changjian Gao and Shih-Lien Lu. Novel FPGA based haar classifier face detection algorithm acceleration. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 373–378. IEEE, 2008.

[31] Michael Hahnle, Frerk Saxen, Matthias Hisung, Ulrich Brunsmann, and Konrad Doll. Fpga-based real-time pedestrian detection on high-resolution images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 629–635. cv-foundation.org, 2013.

[32] M Hasan and A K Roy-Chowdhury. A continuous learning framework for activity recognition using deep hybrid feature models. *IEEE Trans. Multimedia*, 17(11):1909–1922, November 2015.

[33] Daniel Hefenbrock, Jason Oberg, Nhat Tan Nguyen Thanh, Ryan Kastner, and Scott B Baden. Accelerating Viola-Jones face detection to FPGA-Level using GPUs. In *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 11–18. IEEE Computer Society, 2 May 2010.

[34] M Hiromoto and R Miyamoto. Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 894–899. IEEE, 2009.

[35] Zuoxun Hou, Hongbo Zhu, Nanning Zheng, and T Shibata. A single-chip 600-fps real-time action recognition system employing a hardware friendly algorithm. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 762–765. IEEE, June 2014.

[36] Rein-Lien Hsu, M Abdel-Mottaleb, and A K Jain. Face detection in color images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):696–706, May 2002.

[37] Chen Huang and F Vahid. Scalable object detection accelerators on FPGAs using custom design space exploration. In *Application Specific Processors (SASP), 2011 IEEE 9th Symposium on*, pages 115–121. IEEE, June 2011.

[38] G B Huang, V Jain, and E Learned-Miller. Unsupervised joint alignment of complex images. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, October 2007.

[39] Vidit Jain and Erik G Learned-Miller. FDDB: A benchmark for face detection in unconstrained settings. *UMass Amherst Technical Report*, 2010.

[40] Oliver Jesorsky, Klaus J Kirchberg, and Robert W Frischholz. Robust face detection using the hausdorff distance. In *Audio- and Video-Based Biometric Person Authentication*, Lecture Notes in Computer Science, pages 90–95. Springer Berlin Heidelberg, 6 June 2001.

[41] H Jhuang, T Serre, L Wolf, and T Poggio. A biologically inspired system for action recognition. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, October 2007.

[42] Deane B Judd. Hue saturation and lightness of surface colors with chromatic illumination. *J. Opt. Soc. Am., JOSA*, 30(1):2–32, 1 January 1940.

[43] R Kadota, H Sugano, M Hiromoto, H Ochi, R Miyamoto, and Y Nakamura. Hardware architecture for HOG feature extraction. In *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP '09. Fifth International Conference on*, pages 1330–1333. IEEE, 2009.

[44] A Karpathy, G Toderici, S Shetty, T Leung, R Sukthankar, and Li Fei-Fei. Large-Scale video classification with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1725–1732. IEEE, June 2014.

[45] Yan Ke, R Sukthankar, and M Hebert. Efficient visual event detection using volumetric features. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 166–173 Vol. 1. IEEE, October 2005.

[46] Alexander Kläser. *Learning human actions in video*. PhD thesis, Université de Grenoble, 2010.

[47] Alexander Kläser, Marcin Marszałek, and Cordelia Schmid. A spatio-temporal descriptor based on 3D-gradients. In *BMVC 2008-19th British Machine Vision Conference*, pages 275–271. INRIA, 2008.

[48] S Knerr, L Personnaz, and G Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In *Neurocomputing*, NATO ASI Series, pages 41–50. Springer Berlin Heidelberg, 1990.

[49] Ulrich H-G Kreßel. Pairwise classification and support vector machines. In *Advances in kernel methods*, pages 255–268. dl.acm.org, 1999.

[50] Sun Yuan Kung. *Kernel Methods and Machine Learning*. Cambridge University Press, 2014.

[51] I Laptev, M Marszalek, C Schmid, and B Rozenfeld. Learning realistic human actions from movies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[52] Ivan Laptev. On Space-Time interest points. *Int. J. Comput. Vis.*, 64(2-3):107–123, 2005.

[53] Ivan Laptev and Tony Lindeberg. Local descriptors for spatio-temporal recognition. In *Spatial Coherence for Visual Motion Analysis*, Lecture Notes in Computer Science, pages 91–103. Springer Berlin Heidelberg, 2006.

[54] S Lazebnik, C Schmid, and J Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[55] Q V Le, W Y Zou, S Y Yeung, and A Y Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3361–3368. IEEE, June 2011.

[56] D-U Lee, A A Gaffar, R C C Cheung, O Mencer, W Luk, and G A Constantinides. Accuracy-Guaranteed Bit-Width optimization. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 25(10):1990–2000, October 2006.

[57] Seonyoung Lee, Haengseon Son, Jong Chan Choi, and Kyoungwon Min. HOG feature extractor circuit for real-time human and vehicle detection. In *TENCON 2012 - 2012 IEEE Region 10 Conference*, pages 1–5. IEEE, November 2012.

[58] Chao-Tang Li and Wen-Hui Chen. A novel FPGA-based hand gesture recognition system. *JCIT*, 7(9):221–229, 2012.

[59] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Pattern Recognition*, Lecture Notes in Computer Science, pages 297–304. Springer Berlin Heidelberg, 10 September 2003.

[60] Jingen Liu, Jiebo Luo, and M Shah. Recognizing realistic actions from videos “in the wild”. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1996–2003, June 2009.

[61] Jingen Liu, Yang Yang, and M Shah. Learning semantic visual vocabularies using diffusion distance. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 461–468. IEEE, June 2009.

[62] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[63] X Ma, W A Najjar, and A K Roy-Chowdhury. Evaluation and acceleration of High-Throughput Fixed-Point object detection on FPGAs. *IEEE Trans. Circuits Syst. Video Technol.*, PP(99):1–1, 2014.

[64] Xiaoyin Ma, Walid Najjar, and Amit Roy Chowdhury. High-Throughput Fixed-Point object detection on FPGAs. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, pages 107–107. IEEE, May 2014.

[65] Ahmed Al Maashri, Michael Debole, Matthew Cotter, Nandhini Chandramoorthy, Yang Xiao, Vijaykrishnan Narayanan, and Chaitali Chakrabarti. Accelerating neuromorphic vision algorithms for recognition. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 579–584, New York, NY, USA, 2012. ACM.

[66] T Machida and T Naito. GPU & CPU cooperative accelerated pedestrian and vehicle detection. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 506–513. IEEE, November 2011.

[67] Hongying Meng, Michael Freeman, Nick Pears, and Chris Bailey. Real-time human action recognition on an embedded, reconfigurable video processing architecture. *J Real-Time Image Proc*, 3(3):163–176, 14 February 2008.

[68] K Mizuno, Y Terachi, K Takagi, S Izumi, H Kawaguchi, and M Yoshimoto. Architectural study of HOG feature extraction processor for Real-Time object detection. In *Signal Processing Systems (SiPS), 2012 IEEE Workshop on*, pages 197–202. IEEE, October 2012.

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

[70] J Y-H Ng, M Hausknecht, S Vijayanarasimhan, O Vinyals, R Monga, and G Toderici. Beyond short snippets: Deep networks for video classification. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4694–4702, June 2015.

[71] Antonios Oikonomopoulos, Ioannis Patras, and Maja Pantic. Spatiotemporal salient points for visual recognition of human actions. *IEEE Trans. Syst. Man Cybern. B Cybern.*, 36(3):710–719, June 2006.

[72] D Oro, C Fernandez, J R Saeta, X Martorell, and J Hernando. Real-time GPU-based face detection in HD video sequences. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 530–537. IEEE, November 2011.

[73] W G Osborne, R C C Cheung, J Coutinho, W Luk, and O Mencer. Automatic Accuracy-Guaranteed Bit-Width optimization for fixed and Floating-Point systems. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 617–620. IEEE, August 2007.

[74] C P Papageorgiou, M Oren, and T Poggio. A general framework for object detection. In *Computer Vision, 1998. Sixth International Conference on*, pages 555–562. IEEE, January 1998.

[75] Constantine Papageorgiou and Tomaso Poggio. A trainable system for object detection. *Int. J. Comput. Vis.*, 38(1):15–33, 2000.

[76] Ronald Poppe. A survey on vision-based human action recognition. *Image Vis. Comput.*, 28(6):976–990, June 2010.

[77] Victor Prisacariu and Ian Reid. fastHOG-a real-time GPU implementation of HOG. (2310/09).

[78] Huabiao Qin, Lianbing Tian, and Zongwei Hu. A highly parallelized processor for face detection based on haar-like features. In *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, pages 985–988. IEEE, December 2012.

[79] Kishore K Reddy and Mubarak Shah. Recognizing 50 human action categories of web videos. *Mach. Vis. Appl.*, 24(5):971–981, 16 November 2012.

[80] H A Rowley, S Baluja, and T Kanade. Neural network-based face detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(1):23–38, January 1998.

[81] Bernhard Schölkopf, Christopher J C Burges, and Alexander J Smola. *Advances in kernel methods: support vector learning*. MIT press, 1999.

[82] C Schuldt, I Laptev, and B Caputo. Recognizing human actions: a local SVM approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36 Vol.3. IEEE, August 2004.

[83] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the 15th International Conference on Multimedia*, MULTIMEDIA '07, pages 357–360, New York, NY, USA, 2007. ACM.

[84] Yu Shi and Timothy Tsui. An FPGA-Based smart camera for gesture recognition in HCI applications. In *Computer Vision - ACCV 2007*, Lecture Notes in Computer Science, pages 718–727. Springer Berlin Heidelberg, 2007.

[85] Karen Simonyan and Andrew Zisserman. Two-Stream convolutional networks for action recognition in videos. In Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc., 2014.

[86] J Sivic and A Zisserman. Video google: a text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477 vol.2. IEEE, October 2003.

[87] Patrick Sudowe and Bastian Leibe. Efficient use of geometric constraints for Sliding-Window object detection in video. In *Computer Vision Systems*, Lecture Notes in Computer Science, pages 11–20. Springer Berlin Heidelberg, 20 September 2011.

[88] H Sugano, R Miyamoto, and Y Nakamura. Optimized parallel implementation of pedestrian tracking using HOG features on GPU. In *Ph.D. Research in Microelectronics and Electronics (PRIME), 2010 Conference on*, pages 1–4. IEEE, July 2010.

[89] G Sutter and J Deschamps. High speed fixed point dividers for FPGAs. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 448–452. IEEE, August 2009.

[90] C Thurau and V Hlavac. Pose primitive based human action recognition in videos or still images. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, June 2008.

[91] Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2000.

[92] J Villarreal, A Park, W Najjar, and R Halstead. Designing modular hardware accelerators in C with ROCCC 2.0. In *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, pages 127–134. IEEE, May 2010.

[93] P Viola and M Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1. IEEE, 2001.

[94] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *Int. J. Comput. Vis.*, 63(2):153–161, 1 February 2005.

[95] C Wallraven, B Caputo, and A Graf. Recognition with local features: the kernel recipe. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 257–264 vol.1. IEEE, October 2003.

[96] Heng Wang, A Kläser, C Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE, June 2011.

[97] Heng Wang and C Schmid. Action recognition with improved trajectories. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3551–3558. IEEE, December 2013.

[98] Heng Wang, Muhammad Muneeb Ullah, Alexander Klaser, Ivan Laptev, and Cordelia Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC 2009-British Machine Vision Conference*, pages 124–121. hal.inria.fr, 2009.

[99] Jutta Willamowski, Damian Arregui, Gabriella Csurka, Christopher R Dance, and Lixin Fan. Categorizing nine visual classes using local appearance descriptors. *illumination*, 17:21, 2004.

[100] Geert Willems, Tinne Tuytelaars, and Luc Van Gool. An efficient dense and Scale-Invariant Spatio-Temporal interest point detector. In *Computer Vision - ECCV 2008*, Lecture Notes in Computer Science, pages 650–663. Springer Berlin Heidelberg, 2008.

[101] T Wilson, M Glatz, and M Hodlmoser. Pedestrian detection implemented on a fixed-point parallel architecture. In *Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on*, pages 47–51. IEEE, May 2009.

[102] C Wojek, S Walk, and B Schiele. Multi-cue onboard pedestrian detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 794–801. IEEE, June 2009.

[103] Xinxiao Wu, Dong Xu, Lixin Duan, Jiebo Luo, and Yunde Jia. Action recognition using multilevel features and latent structural SVM. *IEEE Trans. Circuits Syst. Video Technol.*, 23(8):1422–1431, 2013.

[104] Chen Yan-ping, Li Shao-zi, and Lin Xian-ming. Fast hog feature computation based on CUDA. In *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, volume 4, pages 748–751. IEEE, June 2011.

[105] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 161–170, New York, NY, USA, 2015. ACM.

[106] Q Zhu, N Garg, Y Tsai, and K Pulli. An energy efficient time-sharing pyramid pipeline for multi-resolution computer vision. In *Very Large Scale Integration (VLSI-SoC), 2013 IFIP/IEEE 21st International Conference on*, pages 278–281. IEEE, October 2013.

[107] Y Zhu, N Nanyak, and A Roy-Chowdhury. Context-Aware activity modeling using hierarchical conditional random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, PP(99):1–1, 2014.