Hardware Acceleration for Multi-Scale Object Detection Based on Dense Pyramid Feature

Congrui Wu[1], Tianmin Rao, Ran Duan, Xiao Zhang[1]

[1] BOE Technology Group Co., Ltd
Keywords: Hardware Acceleration, Object Detection, ACF Detector.

ABSTRACT

ACF is a method for object detection which approximately constructing a dense feature pyramid used for Adaboost classifier. Our work focuses on this method and implement the whole detection process on heterogeneous hardware platform. This design achieves a detection performance of 134 fps consuming less hardware resources.

1 INTRODUCTION

Recent works focus on object detection aims at real-time processing while achieving high performance. A great deal of work has been done to design good detectors and also newly emerged deep learning methods draw much attention. However, the deep learning based method needs complex design and huge resources consumption. This leads to high energy cost and difficult for embedded deployment. For this reason, some hardware implementations focus on traditional image features combined with machine learning and make effect on improving the processing efficiency. In this field, HOG [2] which describe the geometry edge from gradient information draw much of the attention, and plenty of hardware acceleration work such as [3] [4] [5] [6] [7], focus on how to implement this algorithm efficiently and propose simplified methods or other optimized designs.

For multi-scale detection, the HOG-like feature is suffered from the repetitive work on feature calculating of scaled-images. This pixel-level pre-process costs much computation burden and especially makes trouble in hardware design. To optimize this, DollÁr et al. [8] proposal a Fast Feature Pyramid method to reduce complexity and further apply to object detection [1]. Our work utilizes this method and implement a 29-scale dense feature pyramid construction based on pipeline structure on the heterogeneous platform of ZYNQ. The implementation is designed in cost-sensitive and achieve 134 fps for feature construction.

The rest of this work is organized as follows. In Sec.2, the overall of dense feature pyramid construction and ACF based detection flow is introduced. The main idea of complexity reduction and robust objection flow are introduced. In Sec.3, the hardware implementation is shown in the detailed module diagram. In Sec.4 the hardware performance is discussed and the conclusions of this work are presented in Sec.5.

2 ALGORITHM OVERVIEW

The feature pyramid is a set of representations of an image which contains multi-scale information. HOG-like methods construct the pyramid by resizing the original image N times and then compute features respectively. To optimize this, [8] proposes a Fast Feature Pyramid to construct a dense pyramid. In [1], the method is further combined with Adaboost and gains robust detection performance, which is called ACF. A brief introduction of the algorithm is given in this section.

2.1 Fast Feature Pyramid Construction

In traditional HOG-like method, each layer of the feature pyramid is calculate from the resized image of a certain scale. In Fast Feature Pyramid construction, the feature-pyramid is divided into two sub-parts, the base layers and the extension layers. Base-layer feature is calculated from the scaled image data, while extension-layer feature is approximated calculated from neighboring base-layer features reducing the computation to one-third. As shown in Fig.1 and Fig.2 are two different feature construction pipelines of traditional HOG-like method and Fast Feature Pyramid.

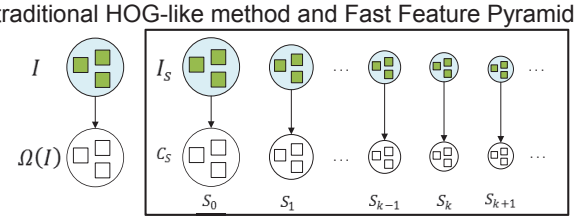

Fig. 1 Traditional method

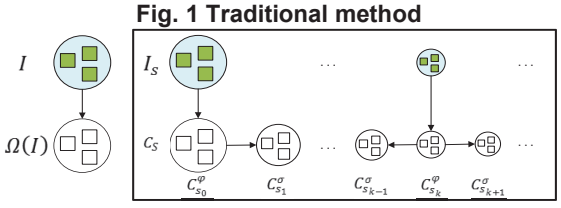

Fig. 2. Fast feature pyramid

This process of fast feature pyramid could be described as below, for a input image I, we define the base-feature C_s^{φ}, a channel of base-layer S_{φ} in the feature pyramid, as

$$C_s^{\varphi} = \Omega(R(I_{\varphi}, s)),$$

And define the extension-feature C_S^{σ}, a channel of extension-layer S_{σ} based on base-layer S_{φ} in the feature pyramid, as

$$C_s^{\sigma} = C_s^{\varphi} \cdot s^{-\lambda_{\Omega}} = \Omega(R(I_{\varphi}, s)) \cdot s^{-\lambda_{\Omega}},$$

Here $s^{-\lambda_{\Omega}}$ is a set of parameters pre-trained for

feature scaling. This approximation avoid the repeating work of feature extracting from all scaled images so that reduce the computation complexity.

After experimental testing, this optimized algorithm is confirmed to be well working and the detection speed is obviously improved at the cost of only a small loss of detection accuracy.

2.2 Object detection flow

The target of Fast Feature Pyramid flow is to construct a dense feature pyramid for object detection. To applying sliding windows for detection, these features are further aggregated. This processing flow is called Aggregate Channel Features (ACF). For object detection, each computed scale contains 10 feature channels. As demonstrated in [8], LUV color information and histogram of oriented gradients (6 channels) with magnitude are concluded. Then, features in each channel are sum by each pixel block accordingly. After vectorizing, the data is applied by Adaboost classifiers for objection detection. The flow is showed in Fig.3.

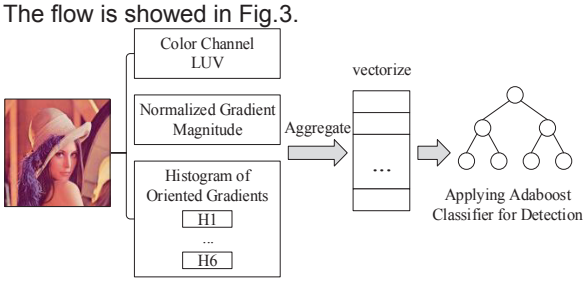

Fig.3 Object detection flow based on ACF

To conclude, the overall detection flow contains two parts: a Fast Feature Pyramid constructed by approximating nearby scales and Adaboost classifiers for a well detection performance.

3 HARDWARE IMPLEMENTATION

In this section, the overall hardware implementation is introduced. To implement the structure described in Sec. 2, a heterogeneous platform called ZYNQ is utilized. This platform integrate both FPGA and ARM processor on a single SoC, of which PL (Programmable Logic) means the FPGA resources on SoC and PS (Processing System) refers to the ARM core on SoC. This structure is suitable for the system aims at high speed pixel-level processing with higher level instruction control.

3.1 Overview of hardware implementation

In hardware implementation, a heterogeneous pipeline architecture is designed to make data flow processing more efficient. As shown in Fig.4, the overall implementation of the system consists of a heterogeneous architecture of PS and PL, of which PL implements feature extraction and PS implements classification detection.

In PL implementation, we firstly calculate the base-layer features C_s^{φ} in parallel structure based on some certain scales (S0, S4, S8, and S16) which are scaled from the input image driven by the source clock. Then extension-layers $C_s^{n\sigma}$ could be calculated in pipeline structure driven by a high frequency clock. In PS implementation, the 29-layer feature is used for object detection with Adaboost classifiers. The parameters pre-trained for Adaboost classifying is loaded into the specific address area in the DDR memory when the system is powered on. PS will receive an interrupt signal after 29-layer feature calculation is completed, then both the parameters and the features would be load into the Adaboost classifiers for detection.

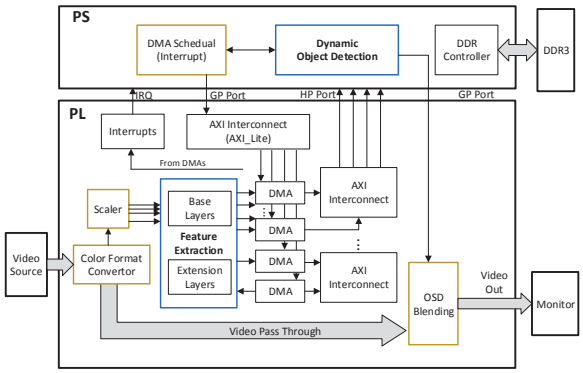

Fig. 4 Block level overview of hardware implementation

3.2 Base-layer feature construction

Fig.5 shows the details of base-layer feature design in the PL implementation. The input RGB image is converted into LUV first. This step is design for obtaining a more robustness feature. Compared with RGB color space, the LUV format is easier to further calculate gradient information. Then, 4 parallel image resizing operation is deployment and 4 base-layer is constructed. The down scalar for 4 base-layer is design to calculate in cascade flow, and both the width and length of each sub-scale is the half of the previous. Thus the model is reused and efficiently gain resized images. After getting resizing results, the “Base-layer Feature Calc” model is responsible to construct the gradient feature channels. Since the grayscale information of the image is fully contained in the “L” channel of the LUV color space, the calculation for gradient data only need to be operated on this data space.

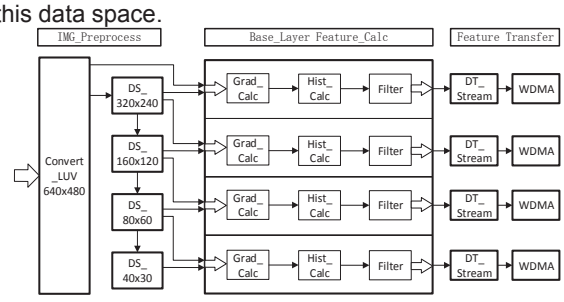

Fig. 5 Base-layer feature construction

3.3 Extension-layer Feature Construction

Fig.6 shows the details of extension-layer feature design in the PL implementation. The resizing for neighboring base-layer feature applied by vertical and

horizontal. In order to reduce hardware resource consumption, this process is designed as a structure for serial computing by module multiplexing. The parameters for scaling is stored in ROM as LUT to speed up the process. After approximating, a smoothing filter is also applied for better performance.

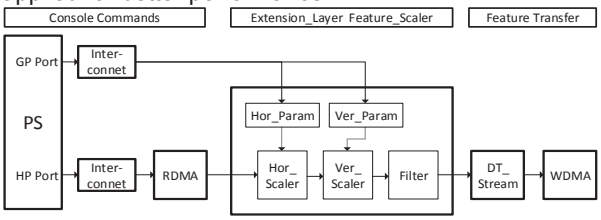

Fig. 6 Extension-layer feature construction

3.4 Objection Detection based on ACF

Hardware implementation of feature pyramid construction generates all feature data needed for detection. Following is the objection detection based on ACF algorithm, which is implemented on the PS sub-part.

In the PS design, to achieve object detection with Adaboost classifier, a sliding window based processing flow is done on each feature layer of the dense feature pyramid containing 10 channels. In this work the size of sliding window is design to be 64x64 for a better performance. The detection result returns the coordinate of a bounding box position along with the score.

4 Evaluation

4.1 Evaluation Platform

The evaluation is based on the Xilinx ZYNQ evaluation board ZC706. The whole platform is show in Fig.7.

2 HDMI cards are connected to transfer 640x480 video and the detection result is showed on the monitor. An UART cable is connected to sending back running information. SD-Card is used for loading Adaboost model and sending back detection result for further evaluation. The evaluation utilizes INRIA data set [9] and the evaluation method.

4.2 Performance

In this section, the performance of this implementation is showed. Since few hardware implementation has been published for Fast Feature Pyramid of ACF, we choose several HOG-like object detection works for comparison to show the performance.

In table 1, the hardware resources and the detection performances are list out. The main contribution of this work is to construct a dense feature pyramid for object detection and save the hardware resource as much as possible. Many other works focus on single-scale processing, such as [3] [6]. To compare these implementations with our work, the hardware reports are equally transferred to show how many resources are utilized for a single-scale processing.

In Fig.8, a demo is showed for this system. By connecting to the video input from PC (a piece of video taken from the road), the result is showed on the screen which indicated by red box.

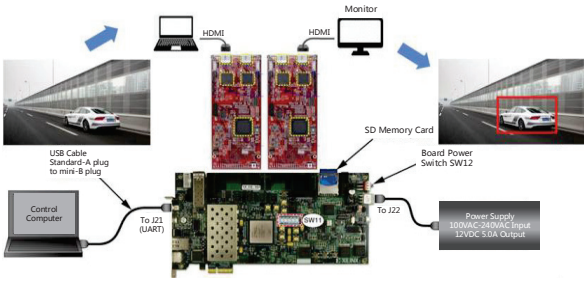

Fig. 7 Hardware test environment

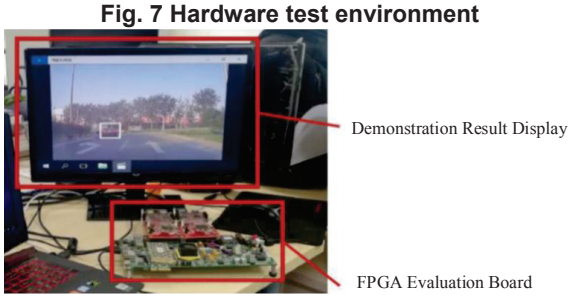

Fig. 8 Real time object detection demonstration

5 CONCLUSIONS

This work accelerates dense feature pyramid construction and followed by detection flow based on ACF. Compared to HOG-like based implementation, the average hardware cost is extremely low due to the nearby scale approximation and the cascade processing on heterogeneous platform ZYNQ. The construction of 29-scale pyramid achieve 134 fps and obtain good detection performance through ACF method. For now, a real-time demonstration could be showed and further of this design is aiming at the potential cost-sensitive application in different scenes.

Table 1 Hardware performance comparison between this work and HOG-like works

Publication	[3]	[4]	[5]	[6]	[7]	This work
Year	2012	2013	2015	2015	2017	2018
FPGA	Cyclone IV	Xilinx Virtex5	Xilinx Zynq	Xilinx Virtex6	Altera Stratix V	Xilinx Zynq
LUTs	34,403	5,188	21,297	98,642	3529	1237
Reg.	23,247	5,178	5,942	8694	2657	1638
DSPs	68	49	4	62	26	8
Clock Rate[MHz]	40	135 + 270	82.2	70 + 140	142 + 284	27 + 200
Resolution	800×600	1920×1080	1920×1080	640×480	1920×1080	640×480
Data rate[fps]	72	64	40	250	68	134
Detection rate	87%	84%	90%	90%	87%	92.4%

REFERENCES

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

[2] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), June 2005, vol. 1, pp. 886–893 vol. 1.

[3] K Mizuno, Y Terachi, K Takagi, and S Izumi, “Architectural study of hog feature extraction processor for real-time object detection,” vol. 52, no. 11, pp. 197–202, 2012.

[4] [4] Michael Hahnle, Frerk Saxen, Matthias Hisung, Ulrich Brunsmann, and Konrad Doll, “FPGA-based real-time pedestrian detection on high-resolution images,” 2013, pp. 629–635.

[5] Jens Rettkowski, Andrew Boutros, and Diana Göhringer, “Real-time pedestrian detection on a XILINX ZYNQ using the HOG algorithm,” in International Conference on Reconfigurable Computing and FPGAs, 2016, pp. 1–8.

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

[7] Jan Drre, Dario Paradzik, and Holger Blume, “A hog-based real-time and multi-scale pedestrian detector demonstration system on FPGA,” in ACM/SIGDA International Symposium, 2018, pp. 163–172.

[8] P. DollÁr, S. Belongie, and P. Perona, “The fastest pedestrian detector in the west,” in BMVC, 2010.

[9] “Inria dataset: http://pascal.inrialpes.fr/data/human/”.