BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

HARDWARE ACCELERATION OF OBJECT DETECTION IN IMAGES

HARDWAROVÁ AKCELERACE DETEKCE OBJEKTŮ V OBRAZE

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **Ing. PETR MUSIL**
AUTOR PRÁCE

SUPERVISOR **Prof. Dr. Ing. PAVEL ZEMČÍK**
ŠKOLITEL

BRNO 2020

Abstract

Nowadays, an increasing number of cameras and surveillance systems can be observed. The amount of information that these devices produce is enormous, and it is not in human power to process it all, therefore using computing power is needed. Modern computer vision algorithms, especially object detection, already achieve excellent results. One of the disadvantages of current vision algorithms is high computational complexity. Therefore, it is desired to implement these algorithms into a suitable device with better performance to power ratio. FPGA represents a reliable option due to its parallel and power-efficient computing. This dissertation aims to propose methods for optimising the object detector in an image running on an FPGA. These detectors use boosted soft cascades of classifiers with local image feature like weak classifiers. The proposed detectors use sequential evaluation of weak classifiers. More positions in the image are evaluated in parallel to increase the detection performance. Also, a new approach for multiscale object detection is proposed; its advantage is no need for external memory. The new detectors were experimentally verified on the tasks of detecting faces and license plates. The results outperform the current state-of-the-art, allow to create object detectors with higher detection performance, better power to resources ratio and better detection accuracy.

Abstrakt

V dnešní době je patrný nárůst počtu kamer a dohledových systémů ve veřejném prostoru. Množství informací které tato zařízení produkují je enormní a není v lidských silách je všechny vyhodnotit a interpretovat. Použití výpočetních technologií je nezbytné. Moderní algoritmy počítačového vidění již dosahují skvělých výsledků, jejich širšímu použití v praxi zatím brání nízký výkon zařízení a vysoké požadavky na výpočetní zdroje a energii. Jednou z možností je využití vysokého paraelního výkonu FPGA pro efektivní zpracování těchto algoritmů. Cílem této disertační práce je představit navržené metody optimalizace detektoru objektů v obraze běžících na FPGA. Tyto detektory využívají boostovatelné soft kaskády klasifikátorů spolu s lokálními obrazovými příznaky, které slouží jako slabé klasifikátory. Navržené postupy využívají sekvenční vyhodnocení slabých klasifikátoru. Pro zvýšení výkonu detekce je vyhodnocováno současně více pozic v obraze. Je navržen nový přístup pro detekci objektů různé velikosti nevyžadující externí paměť. Vytvořené detektory byly experimentálně ověřeny na úlohách detekce obličejů a poznávacích značek automobilů. Dosažená výsledky překonávají současný stav poznání, umožňují vytvořit detektory objektů s vyšším detekčním výkonem, lepším poměrem výkonu a spotřebovaných zdrojů FPGA a s lepší přesností detekce.

Keywords

Object Detection, AdaBoost, WaldBoost, Acceleration, FPGA

Klíčová slova

Detekce objektů, AdaBoost, WaldBoost, Akcelerace, FPGA

Reference

MUSIL, Petr. *Hardware acceleration of object detection in images.* Brno, 2020. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Prof. Dr. Ing. Pavel Zemčík

Hardware acceleration of object detection in images

Declaration

Hereby I declare that this dissertation thesis is my original work and that I have written it under lead of Prof. Dr. Ing. Pavel Zemčík. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Petr Musil
August 30, 2020

Acknowledgements

I would like to acknowledge and thank the people who have supported me, not only during my doctoral studies. First of all, to Pavel Zemčík, my supervisor, for his guidance, knowledge and sustained support. To my colleagues and friends for their personal and professional support, especially to Michal Hradiš, Roman Juránek and Martin Musil. I would also like to thank my family for their patience and my girlfriend, Karolina for language corrections, and her support and love.

Contents

Glossary

ACF Aggregated Channel Features.

BRAM Block Random Access Memory element on FPGA.

CLK FPGA clock speed in MHz.

CNN Convolution Natural Network.

DSP Digital Signal Processing element on FPGA.

FF Flip Flop register on FPGA.

FPGA Field-Programmable Gate Array.

FPS Frame Peer Second.

HSG Histogram of Significant Gradients.

LBP Local Binary Patterns.

LRD Local Rank Difference.

LUT Look Up Table on FPGA.

SVM Support Vector Machine.

Chapter 1

Introduction

Nowadays, an increasing number of cameras and surveillance systems can be observed. We can see cameras at toll gates, security cameras in buildings or police surveillance systems. The amount of information that these devices produce is enormous, and it is not in human power to process and interpret it all. The only option is to use computing power to analyse the huge number of videos and frame sequences. Modern computer vision algorithms have passed the point, where it is reasonable to start implementing them widely. Algorithms for object detection and recognition, for example of human faces, pedestrians, cars, or traffic signs already outperform human. In general, one of the disadvantages of advanced vision algorithms is high computational complexity. For this reason, it is necessary to use powerful computer systems with high energy consumption and cost. Also, it would be convenient to process most data locally without the need for remote servers, so-called edge computing.

One solution can be the hardware acceleration of computer vision algorithms on FPGA or ASIC chips. The aim is to create low-cost, low-power devices for real-time video processing. The deployment of algorithms to FPGA and ASIC circuits is specific and differs greatly from deployment to conventional computing systems. Usually, a direct implementation of computer vision algorithms without their modification is inefficient, slow and resource-intensive. The acceleration of computer vision algorithms in hardware has long been the goal of many scientific works. This topic is attractive due to its potential practical application, and a combination of different research fields: image processing and hardware acceleration.

This thesis presents my contributions to the state-of-the-art in the topic of visual object detection in FPGA. Specifically, the work is focused on fast and powerful object detectors with low demands on resources. Such detectors could be applied mainly in transport, industry or security. One of the applications of the detector will be demonstrated on the task of detecting license plates for parking control in residential zones. The benefits will be shown in comparison to current technologies.

The contribution itself is in proposing methods for optimizing object detection on FPGAs. The main focus is on detectors using boosted soft cascades of classifiers with local image features as weak classifiers. Sequential evaluation of weak classifiers has been upgraded with parallelization by evaluation of several independent image positions simultaneously. Also, a new approach for multi-scale object detection has been proposed; its advantage is no need for external memory. Using these methods to create effective detector verifies the hypothesis: that it is possible to design an object detector based on soft cascade deployed in programmable hardware with resulting precision comparable to the state-of-the-art, with

real-time performance, with lower power consumption and less computing resource demands comparing to existing ones.

The thesis consists of a commented set of articles. Next chapter introduces related state-of-the-art. Chapter 3 presents the main contribution of the work. Chapters 4,5 and 6 are core of my dissertation thesis, representing commented re-formatted copies of my papers [1, 2, 3]. The chapter 7 describes the application of the detector into practice, using it in several research projects and outlines future work. The work finishes with a conclusion.

Chapter 2

Object detection in images using embedded devices

This part of the dissertation thesis provides an overview of the state of the art of object detection on hardware platforms. The focus is mainly on general object detection using boosted classifiers and on a summary of other authors' work on the topic of detection in hardware. This chapter is included to introduce the reader into the topic because the papers I published did not offer enough space for more detailed information. Recent scientific contributions on the topic are also described, as our articles were published through the years 2013-2020, and the state-of-the-art has evolved since then.

Object detection in image is one of the fundamental algorithms of computer vision. The definition of object varies and it is largely application dependent. It is often defined by a set of annotated example images [4]. Object detection in image is a popular topic in the scientific community with wide practical scope. Over the years, many approaches to object detection have been proposed. From the first methods based on hand-designed ad-hoc detectors [5, 6] or template matching [7] trough part-based methods [8, 9, 10] the field progressed to appearance-based detectors [11, 12, 13, 14, 15, 16, 17, 18].

The appearance-based detectors use statistical analysis and supervised machine learning methods to learn distinctive object characteristics. The first of these methods used Support Vector Machines (SVM)[19] in combination with Haar's features[20] or Gabor filters[21]. SVM is a mathematical method for searching best separating hyperplane in a feature space. It produces classifiers with a simple structure suitable for parallel implementation. Modern approaches use SVM and Histograms of Oriented Gradient(HOG) for detection of pedestrians[22, 23, 24, 25, 26], cars[27], traffic sign[28], etc. The most successful detection methods designed for devices with limited computational resources are based on sliding windows and boosted classifiers [11, 12, 13, 14, 29, 30, 31, 32, 33, 34, 35]. These methods enable creating powerful and accurate universal detectors of relatively rigid and visually distinct objects such as faces [11, 36, 33, 30], pedestrians[14], traffic signs[37], licence plates[38, 39], etc. Another large class of appearance-based detectors builds on deep learning, specifically on Convolutions Neural Networks (CNN) [40]. These methods represent state-of-the-art in terms of detection accuracy and variability of objects they can handle [15, 16, 17, 41].

Over the years, many hardware implementations of various object detectors have been proposed. The implementations typically belong to following categories of detection methods:

- Ada-Boost based detectors – cascades of boosted classifiers [11] with Haar image features [29, 30, 31, 32, 33, 34, 35] or soft-cascades [12] typically with LBP/LRD features [42, 43, 44].
- SVM [19] typically in combination with HOG [23, 24, 25, 27, 28].
- Boosted decision tree with Aggregate channel feature(ACF) detector [45, 46].
- CNN in various forms [47, 48, 49, 50, 51].
- Other methods implementing detection using background subtraction [52], keypoints [53] or ad-hoc detection algorithms [54].

2.1 Object detection using boosted classifiers

Viola and Jones [11] in 2001 presented the first practical general object detector. It it uses an efficient cascade of boosted classifiers with Haar-like image features (weak classifiers) in a sliding window fashion to gradually classify overlapping image windows into background and object classes. Viola and Jones used the learning algorithm Adaptive Boosting (AdaBoost) [55] which selects and sorts weak classifiers each based on a single computationally simple image feature by their importance. Haar-like features encode local image freqency and can be efficiently calculated using integral image in constant time. The main advantage of the detector is the use of the attentional cascade of classifiers, which is a mechanism that decides very quickly on simple background areas and spends more time at ambiguous positions. This ultimately reduces the computational complexity of detection by several orders of magnitude. The combination of efficient features, powerfull classifiers and the cascade structure resulted in a first real-time detector of the frontal face (running at 15 frames per second on 384x288 pixel images at the time of publication). The Viola-Jones detector had gained immense popularity and was the basis to a significant number of modifications [12, 13, 56, 57, 14].

The most effective modifications of the attentional cascade of Viola-Jones is to let the individual cascade stages to share information and to increase the number of rejection decisions. For example, the soft-cascade of Bourdev and Brand [12] produces one long classifier and thresholds for rejection decisions after each weak classifier. The rejections in the Viola and Jones detector occur after the end of each stage, one of which usually contains tens of weak classifiers. The combination of more frequent rejection decisions and a single continuous classifier results in much faster detectors while maintaining the same accuracy. However, the threshold selection method proposed by Bourdev and Brand is far from optimal.

Šochman and Matas [13] proposed optimal rejection threshold selection scheme for soft-cascades inspired by Wald's Sequential Probability Ratio Test (SPRT). They used SPRT to generate an optimal sequential decision strategy on weak classifiers selected by AdaBoost which can additionaly include also positive acceptance thresholds. The resulting WaldBoost algorithm can be considered the state-of-the-art in the field of boosted detectors.

Boosted classifier used for object detection can be build on various types of week classifiers and image features. The main requirements on weak classifiers are high discriminative power and low computational complexity. Viola and Jones [11] and follow-up works [12, 29, 30, 31, 32, 33] utilised Haar-like image features. Haar features are wavelet features that extract local frequency information. They originated from the theoretical

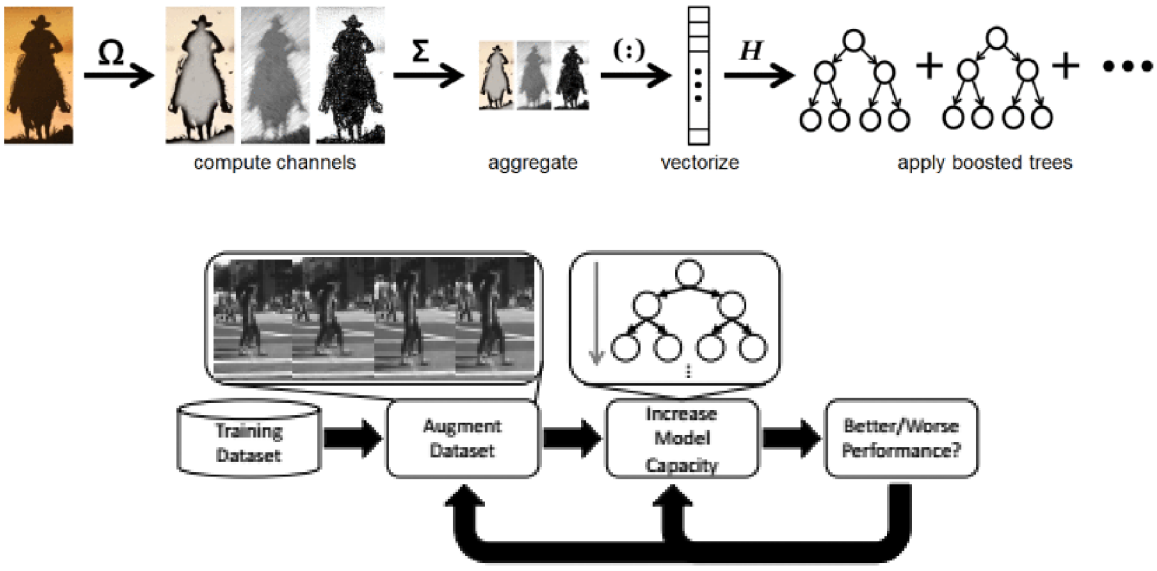

Figure 2.1: (Top) ACF detector principle [57]. (Bottom) The principle of boosted cascade training optimization as suggested by Bar et al. [14]

work by A. Haar [58]. Haar wavelets respond to oriented edges and bars in images and Haar features can be calculated by convolution of an input image with the Haar wavelets. The advantage of Haar features is constant time calculation from an integral representation of an image.

Very popular features for object detection [42, 43, 59] are local binary patterns (LBP) [60] which capture local shape of image intensity. They use sampling of the local neighbourhood to construct a binary code from intensity values which is invariant to monotonous changes in image intensity and can be calculated in constant time. Similar features named Local Rank Differences (LRD) [61] and Local Rank Patterns (LRP) [62] were proposed to reduce memory requirements for weak classifier coefficient and for effective implementation on GPUs [63, 59] and FPGA [64].

Dollar et al. [56] suggested to combine more types of image features as precomputed image channels to improve detection accuracy (named aggregated channel features - ACF). They use a combination of gradient histograms (HOG), colour (including grayscale, RGB, HSV, and LUV), and gradient magnitude. For better performance, the features are precalculated to separate image channels and optionally aggregated to lower resolution. Dollar et al. also proposed simple decision trees as weak classifiers in the soft-cascade. ACF was further extended by an effective multi-scale detection using Fast Feature Pyramids [57] which calculates feature channels only for a sparse set of image scales which are used to efficiently approximate the rest of the scales. This saves computing power and possibly memory of hardware platforms.

Ohn-Bar and Trivedi [14] focused on the limitations of the boosted classifiers and described the relationship between the capacity of boosting classifier, dataset size, and dataset properties. They introduced knowledge from neural network training as data augmentation into the training of boosting classifiers. Besides, they inspected the effect of increasing model capacity on accuracy. They demonstrated that combining these approaches improves the accuracy of detection. The resulting detectors (ACF+ and LDCF+) provide the best-known accuracy among non-CNN techniques while operating in real-time.

2.2 Ada-Boost based hardware detectors

Since Viola and Jones published their real-time detector [11], there has been much effort to implement the detector in hardware, typically in FPGAs [29, 30, 31, 32, 33, 34, 35, 42, 43, 44, 65].

One of the first FPGA implementations was proposed by Lai et al. [29]. Their parallel implementation of the original Viola and Jones detector achieved a speeds up to 143 frames per second (FPS) at 640x480 resolution and a single scale. Due to high demands on FPGA resources, they had to limit the cascade to only the first three stages (52 features), which led to low detection accuracy and the other stages had to be computed on CPU. Cho et al. [30] proposed a similar approach with several parallel blocks computing classifiers at different locations to accelerate the processing speed. Their implementation supports multi-scale detection at the cost of storing the whole image to FPGA memory. This leads to high memory demands and limited image resolution. These implementations of the original detector used Haar feature with an integral image which is not really suitable for FPGAs for several reasons. The integral image increases memory requirements — each pixel requires a higher bit depth depending on the Haar feature size (even 20 and more bits per pixel are need). However, the calculation of the features without the integral image directly from the image data is very computationally demanding and not possible in constant time. The integral image allows calculation in constant time and the maximum number of memory reads required is 9 from different parts of the detection window depending on the specific Haar feature shape and position. Such memory reads can not be aligned and result in inefficient memory access. Reading these values from BRAM is restricted by the number of memory ports and the parallelization is limited by the non-uniform memory access pattern. Typicaly, the memory access limitation is mitigated by scaling the image and implementing the sliding window as a register array with FIFO line buffers (stored in BRAM) to enable fast reading of integral image in a single clock cycle. It allows for parallel access to all pixels in the window using a multiplexer network. The size of this multiplexer network increases linearly with the size of the detection window and with the number of pixels accessed in parallel. This causes enormous demands on logic resources and limits the implementations to using only small and fixed window sizes.

Huang and Vahid [34] partially solved the multiplexer network size problem by limiting feature positions. They developed a method with automatically generates a minimal multiplexer network for a specific detector. Brousseau and Rose [35] reduced the multiplexer network size by preloading adjacent pixels, allowing parallel evaluation of classifiers in adjacent scanning windows. However, this requires the use of a very complex evaluation control mechanism which is necessary to rearrange execution of classifiers after some are terminated in order to maintain high utilization of the parallel memory accesses.

Other works used local image patterns instead of Haar features. Jin et al. [42] proposed a design of a fully pipelined monolithic Ada-boost classifier with LBP which executes a all features for one detection window position in parallel per one clock cycle. This results in a high-speed detector; however, the demands on logic and register resources are enormous. Kadlcek and Fučik [43] proposed similar fully pipelined architecture with LBP features utilizing unique and unusual LBP shapes selected by a genetic algorithm. However, the high expense of FPGA resources allows only for the implementation of a limited number of weak classifiers.

Zemčik and Žadnik [44] suggested an approach based on the Wald-Boost detection algorithm with local rank differences (LRD) features. They precompute and store several

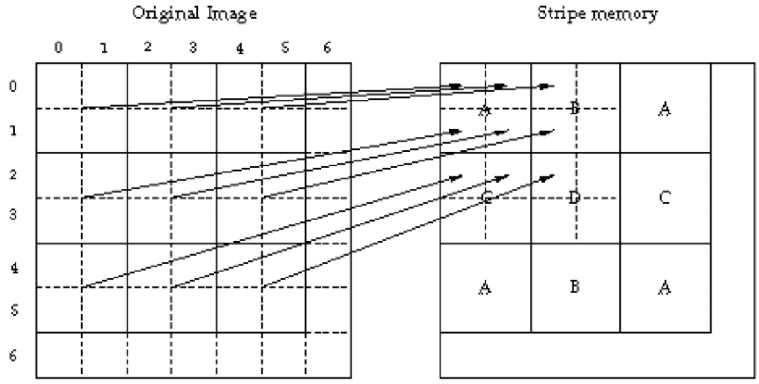

Figure 2.2: Complex memory access structure for LRD feature evaluating proposed by Zemcik[44].

Table 2.1: Comparison of performance and power consumption of Ada-Boost based hardware detectors. *Only a pre-dection with a shortened classifier is done in FPGA, further post-processing is needed.

	Feature	Image size	FPGA	LUT	FF	BRAM	CLK	FPS
Lai 2007* [29]	Haar	640×480	Virtex2 VP30	21K	8K	44	126	143
Granat 2007 [31]	Haar	256×256	Virtex2 LX250	–	–	100	24	< 5
Zemcik 2007 [44]	LRD	640×480	Virtex-II	1490(SL)	–	14	100	22
Hiromoto 2008 [32]	Haar	640×480	Virtex5 LX330	63K	56K	–	160	30
Cho 2009 [30]	Haar	640×480	Virtex5 LX110T	67K	22K	41	–	7
Kyrkou 2011 [33]	Haar	320×240	Virtex2 VP30	26K	24K	24	100	64
Huang 2011 [34]	Haar	320×240	Virtex5 LX155T	80K	–	–	65	100
Brouss 2012 [35]	Haar	320×240	Stratix4 GX530	–	–	–	125	50
Jin 2012* [42]	LBP	640×480	Virtex5 LX330	128K	75K	286	125	300
Kadlcek 2013* [43]	LBP	1024×1024	Virtex2 LX250	1007(SL)	–	31	130	130

smoothed images corresponding each to a different shape of LRD features. A sophisticated memory pattern allows reading block of 3x3 values to evaluate one weak classifier in one clock cycle. However, storing the precomputed values increases demands on memory, and the complex memory access structure requires too many logic resources.

2.3 ACF based hardware detectors

Object detectors based on ACF [57] are very popular due to their excellent performance, good accuracy, and an available set of ready-to-use classifiers (faces, pedestrians, traffic signs, cars) and a toolkit to train user-specific classifiers [66]. Song et al. [67] proposed the first implementation of ACF in FPGA for pedestrian detection in the driver assistance system. This non-parallel implementation of the original ACF detector used ten feature channels. Multi-scale detection, however, requires multiple readings of the image from an external memory.

Mitsunari et al. [45] introduced a more effective implementation of ACF. They focused on problematic parts of the algorithm to enable parallelization on FPGA. The calculation of the HOG feature (using trigonometric functions and a square root) was replaced by an approximation (using only multiplication and addition). Further, they reduced memory requirements with only 2% accuracy penalty by quantizing both classifier's coefficient and thresholds (from 32 to 2 bits). Decision trees used in ACF as weak classifiers requires

Table 2.2: Comparison of performance and power consumption of ACF and SVM based hardware detectors. *Only feature preprocessing done in FPGA. **Only a pre-dection with a shortened classifie

	Type	Stride	Image size	FPGA	LUT	FF	BRAM	DSP	CLK	FPS
Martelli 2011* [68]	SVM+Covariance	8	640×480	XC6VLX240T	1553 (SL)	–	3	22	154	132
Yazawa 2015 [69]	SVM+HOG	5	640×480	CycloneIII	17K (LE)	11K	–	–	70	13
Ma 2015 [23]	SVM+HOG	4	1620×1200	XC6VLX760	46K	187K	381	190	150	10
Said 2016* [70]	SVM+Covariance	4	640×480	XC6VLX240T	1357 (SL)	–	8	46	222	292
Song 2016 [67]	ACF	4	640×480	–	–	–	–	–	166	30
Kyrkou 2016 [65]	SVM+LBP	5	800×600	Spartan6 LX150T	33K	20K	256	59	70	40
Bilal 2017 [71]	SVM+HSG	4	640×480	Cyclone IV	751	496	3	0	50	25
Mitsunari 2018 [45]	ACF	4	1920×1080	ZX7Z045	138K	149K	389	128	–	176
Durre 2018 [25]	SVM+HOG	8	1920×1080	Stratix V	3529	2657	~41	26	142	68
Wang 2018* [24]	SVM+HOG	4	640×480	Cyclone IV	17K	7K	338	144	108	60
An 2019 [72]	SVM+HOG	8	1920×1080	Stratix IV	7625	4503	~7	41	–	60
Li 2019** [73]	SVM+HOG	8	512×512	Stratix IV	313K	90K	859	268	320	10000

complicated memory access because selected decision nodes depend on the input data. Thus, parallel processing is complicated due to memory access conflicts. Mitsunari et al. resolved this issue by storing each channel in a separate memory bank in combination with SIMD-like processing which enabled channel-wise parallel implementation. To maximize the efficacy, the memory access conflicts are minimized by a complex processing order scheduling.

2.4 SVM based hardware detectors

SVM is a prevalent classification algorithm utilized for implementing object detectors in FPGA [23, 68, 69, 70, 65, 71, 25, 72, 73]. This is thanks to the fact that the SVM detectors have simple rigid structure suitable for parallel implementation — they include space-uniform feature extraction and a multi-channel convolution. SVM classifiers with HOG features [22] have support in OpenCV [74] which also includes set of pre-trained classifiers (pedestrians, traffic signs, cars, etc.). This facilitates experimentation and testing. However, the absence of an attention method in the basic SVM detectors is disadvantageous. Unlike boosted detectors, SVM detectors do not incorporate early rejection, therefore it is necessary to evaluate all features at all image positions. That leads to increased demands on computing resources and/or processing time with the associated increased power consumption. HOG feature evaluation on FPGA requires a large number of complex computations on floating-point arithmetic. Square root, arctangent, and normalization (division) evaluations are necessary. [75]

Martelli et al. [68], and Said and Atri [70] proposed a SVM detector with set of image features extracted by learned linear filters. The calculation of the above is much easier than HOG evaluation. They use FPGA to accelerate the feature extraction. The SVM calculation itself takes place in a connected general purpose processor.

Ma et al. [23] implemented the Opencv version of the HOG+SVM detector [22]. They applied a fixed-point arithmetic instead of the original floating-point arithmetic. In their implementation, HOG feature rows are processed in parallel. Since cells in one row can be used for block normalization of the next row, alternating between odd and even rows prevents computing histograms twice, and leads to a processing speed-up. Bilal et al. [71] use Histogram of significant gradients (HSG) instead of HOG. The hardware for HSG calculations is simplified and does not require floating-point arithmetic.

Table 2.3: Comparison of performance and power consumption of selected CNN based hardware detectors

	CNN	Image size	FPGA	LUT	FF	BRAM	DSP	CLK	FPS
Nakahara 2018 [47]	Light YOLOv2	224×224	XCZU9EG	135K	370K	1706	377	300	300
Ma 2018 [48]	SSD	300×300	GX2800	532K	–	3844	4363	300	34
Nguyen 2019 [49]	Simple YOLOv2	416×416	XC7VC707	155K	115K	1144	272	200	60
Kang 2019 [50]	VGG16+SSD	640×480	XC7VX690	181K	497K	1470	3074	210	42
Wu 2019 [51]	MobileNet+SSD	–	XCZU2	161K	301K	771	2070	430	31

Li et al. [73] suggested a high-speed vision platform for detection of multiple highly distinctive objects. They use a short SVM classifier with HOG features in a highly parallel system that receives the input of 64 pixels per clock cycle. This allows detection of fast-moving objects at 10000 fps.

2.5 CNN based hardware detectors

Historically, the first hardware object detectors used [76] neural networks. They were typically designed to handle one specific problem and did not achieve satisfactory performance and accuracy. The use of neural networks was then temporarily abandoned, and the focus was on methods based on boosted classifiers. Along with the growing popularity of deep learning for object detection [15, 16, 17], a large number of articles discussed the possibilities of their practical implementation on hardware platforms [77, 78, 49, 51, 47].

CNN are computationally intensive. Graphic Processing Units (GPUs), which have massive parallel performance, enable to compute CNN-based detectors in a reasonable time. However, GPUs have very high power consumption, so the use of FPGAs could have significant benefits due to massive parallel and power-efficient computing. However, it is hard to deploy standard neural networks into embedded devices because of a large number of operations and parameters CNN-based detectors have. Another disadvantage is the use of floating-point arithmetic in standard networks, which is resource-intensive on FPGAs.

Several early FPGA implementations used the floating-point representation that has enormous computation costs [77, 78]. Some works [83, 84, 85] demonstrated that a floating-point representation is unnecessarily redundant and the CNNs coefficients and intermediate results can be retrained and quantized to a very low-bit precision (even 1 or 2 bits for network weights) without a significant loss of accuracy. The quantization approach has been adopted for FPGAs by multiple authors [49, 51, 47]. Nakahara et al. [47] used standard YOLOv2 CNN [16] and implemented a mixed-precision CNN, which consists of binarized input layers and half-precision (16 bit) output CNN layers. The resulting detector achieved better accuracy than a fully quantized detector. Nguyen et al. [49] used a binarized version of YOLO CNN [15] and found that memory access and memory throughput to external DRAM memory is the main factor limiting performance. They focused on decreased dynamic random access to memory in order to increase performance.

Another approach how to optimize CNN-based detector is to use lightweight architectures such as Xception [18] and MobileNetV2 [17] which use depthwise separable convolutions as a replacement to the standard convolutions. Depthwise separable convolutions significantly reduce the number of operations and parameters with only a limited loss of accuracy. Wu et al. [51] compared standard convolution with separable convolution, and in addition, they rearranged input features and weights to increase performance.

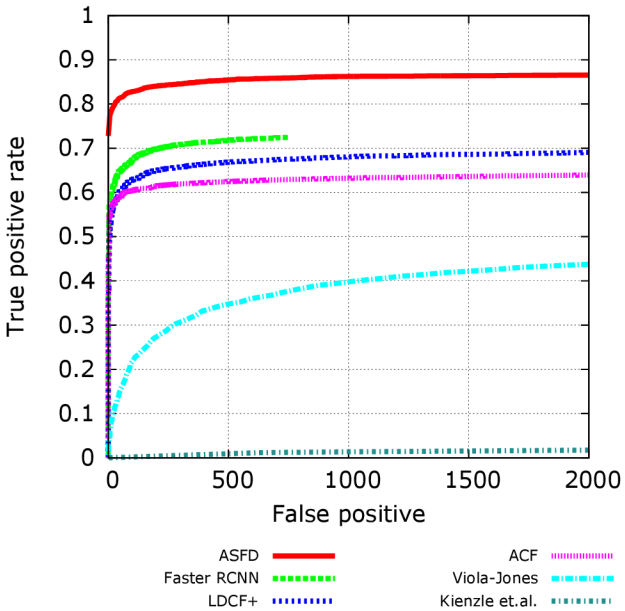

Figure 2.3: ROC curves of selected detectors on FDDB: A Benchmark for Face Detection in Unconstrained Settings [79]. ASFD [80] - the best current method using CNN; Faster RCNN [41] - very popular CNN based method; LDCF+ [14] - the best current boosted classifier based method(ACF+); ACF [57] - boosted decision tree classifier; Viola-Jones [11] - original algorithm Viola-Jones using AdaBoost and Haar features; Kienzle et. al. [81] - SVM based detector with HOG features

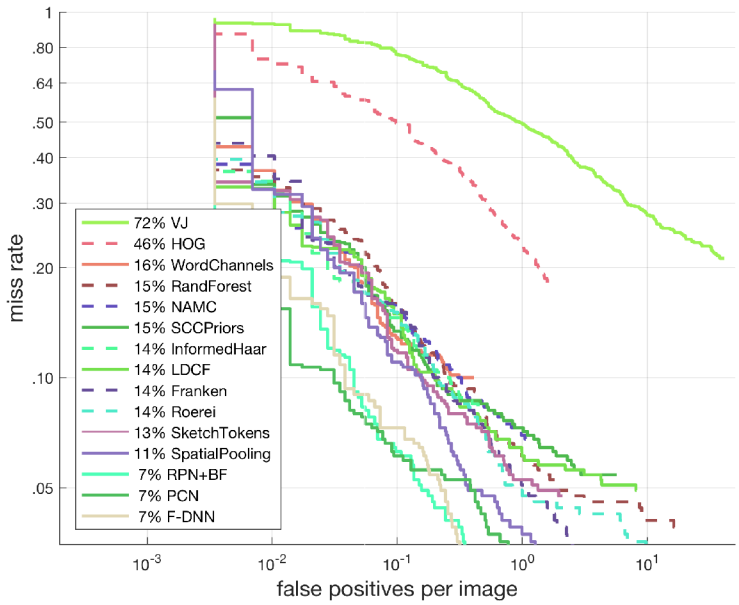

Figure 2.4: ROC curves of selected detectors on Caltech Pedestrian Dataset [82]. Important algorithms: VJ [11] - original Viola-Jones detector using AdaBoost and Haar features; HOG [22] - SVM based detector with HOG features; LDCF [57] - boosted decision tree classifier with ACF; F-DNN [26] - the best current CNN based method

2.6 Summary

Figure 2.3 and Figure 2.4 illustrate a comparison of selected types of detectors for faces [79] and pedestrians [82]. These datasets are standardly used to benchmark detection methods and they include highly challenging scenarios including partially overlapped objects, over- and underexposure, tiny and big objects, and various rotations. Comparisons on older datasets (as used by Viola and Jones [11]) would make no practical sense nowadays because CNN-based methods achieve nearly 100% accuracy at the moment. The ROC curves show that algorithms based on CNN achieve the best accuracy in both detection tasks. Advanced boosted algorithms ACF [57] and ACF+ [14]) provide a slightly lower accuracy but still outperform the original algorithms by Viola-Jones and SVM+HOG based [26, 22]. This is especially evident in face detection, where SVM+HOG based algorithms practically do not work at all. However, these are the results of standard implementations running on a CPU or GPU. Efficient CNN detector implementations in hardware use optimization methods such as quantization of coefficients or network size limitation which, in general, reduce detection accuracy [83, 84] almost to the level of the boosted classifier based algorithms.

Overview Tables 2.1, Tables 2.2 and Tables 2.3 display the speed and amount of resources required for various implementations of the object detector in hardware. SVM based detectors achieve seemingly excellent performance; for example, Li [73] reports 10000 fps at resolution 512x512 pixels. However, direct comparison of the reported frame-rates is almost meaningless as the detectors are designed for different types of objects, they provide different detection accuracy, the scan images at different resolutions and with different window strides. Moreover, the individual detectors significantly differ in the resources and power they consume. In general, modern boosted classifiers provide very good accuracy and speed trade-off in high-throughput and resource limited scenarios, where only a small number of object classes with relatively consistent appearance need to be detected. CNNs excel at more complex detection tasks where resource and power effectiveness is not that crucial.

Chapter 3

Goals and Contributions

This thesis focuses on object detection in images on hardware platforms. The scope of the work is to shift the scientific knowledge and apply it into the practice.

Presumed usage of object detectors developed here is industrial, transport or security applications, i.e. in tasks such as the detection of faces, pedestrians, products, licence plates. Practical deployment of such object detectors needs to meet specific requirements. The resolution of the processed image is an important parameter. This resolution depends a) on the resolution of the input image, b) on the expected size ranges of the searched objects and c) on the size of the detection window. Thus, the expected object size ranges significantly affect the required performance of the detector. Traffic and security applications require approximately 10 to 20 frames per second for suitable object tracking. Industrial applications often require even higher processing speeds. High-speed detectors also allow for processing image data in the camera without storing them on a fast external memory; the absence of external memory further reduces the price of the resulting device.

A typical requirement is that the detectors should achieve the best accuracy possible. In general, object detectors often balance a speed-precision trade-off. Detection of visually diverse, rotated, or distorted objects will either be less accurate or will require a complex system with a large number of computational resources. For example, in the task of licence plates detection in toll gates, where the approximate size and rotation is known, excellent accuracy can be expected. However, a similar task, licence plates detection for parking control in residential zones, is more challenging due to unconstrained conditions (variable position, scale, rotation, etc.), and therefore lower accuracy can be expected.

The technical goal is to create a powerful universal object detector for FPGA hardware with good accuracy and low resource consumption. Such detector should process at least FullHD video at 15 frames per second and should detect small and large objects with limited variability. The detector should achieve accuracy (recall with precision 0.70) of at least 95% for face detection and 99% for LP detection. Important parameters for the low price of the device are low power consumption and a small number of FPGA resources used. The detectors are expected to be utilised in smart cameras. Such smart camera should perform the maximum number of operations directly in-site and send out only the results for further processing. Thanks to this, it would be possible to reduce the data flow from the camera as expected by edge computing.

3.1 Technical implications

At the beginning of my research work, there were already several successful attempts to create an object detector in hardware [30, 31, 32, 33, 34, 35, 42, 44]. A detailed overview of the individual solutions was given in the previous chapter. In summary, the results show that the practical use of these detectors is limited. In order to increase the performance and accuracy of detectors and the resolution of the processed image, it is necessary to improve current detection methods in hardware, modify detection algorithms and apply hardware-specific features.

Choice of classifier algorithm

Hardware detectors based on boosted classifiers have the most advantages for the applications mentioned above. The specifically targeted ad-hoc detectors have an excellent power-to-resource ratio in some applications. However, the ad-hoc detectors are not universal, the accuracy of detection is low, and they do not cope with changing conditions. Creating a detector for a new object class means a lot of work and an uncertain result. The SVM based detectors have good performance and relatively low resource consumption. However, they provide low detection accuracy not applicable in modern applications.

CNN based detectors provide the best accuracy and excel at complex detection task. Due to a large number of operations, they have high demands on logic and memory resources and have relatively low performance. Besides, the CNN optimisations for FPGAs means a loss of accuracy down to the level of a modern boosted classifier. Considered tasks such as face/pedestrian detection and licence plate detection have only a small number of object classes with a relatively consistent appearance, for which CNN seems unnecessarily complex. In conclusion, the modern boosted classifiers provide sufficient accuracy and speed trade-off considering these tasks with high-throughput and resource-limited scenarios.

Individual modifications of boosted classifiers, such as soft-cascade or Waldboost [11, 12, 13], only differ in the training process. The evaluation step does not vary much; the only difference is that soft-cascades and Waldboost allow rejecting after each weak classifier and the original Viola-Jones algorithm [11] allows rejecting after each stage (a set of several weak classifiers). The planned detector should evaluate all modifications of boosted classifiers. The best way to reach high detection accuracy and performance is training the boosted classifiers with Waldboost algorithm [13] and with augmentation, as suggested by Bar et al. [14].

Two approaches for implementation of boosted classifier based detector were developed — fully pipelined monolithic detector [42, 43] and sequential detector [44, 35, 33]. Fully pipelined detectors assess all features for one detection window position in parallel per one clock cycle. In general, the fully pipelined detectors are high-throughput and easy to implement. However, they have high demands on FPGA resources and usually evaluate only a limited number of weak classifiers, which leads to lower accuracy. Sequential detectors assess the features gradually. The parallelisation of sequential detectors is possible by processing multiple features or multiple positions at the same moment. They require fewer resources and allow for better accuracy, but they are more challenging to control. I have focused on sequential detectors because they meet the required parameters better and offer space for further development.

Choice of weak classifiers

Many authors [31, 32, 30, 33, 34, 35] use Haar-like features as weak classifiers. High bit depth for storing integral image and complex logic to access feature values make them resource demanding. Therefore, it isn't easy to increase the performance of such detectors. Other works [44, 42, 43] apply LBP or their modifications (LRD, LRP) as weak classifiers. Their advantage is in loading only surrounding pixels (block reading) for effective calculation. LBP-like features are mainly used in combination with fully pipelined detectors [42, 43]. Zemčík and Žádník [44] verified the sequential approach by developing a suitable object detector. Their detector used LRD features with a size of 3x3 and subsampling of the original image in the ratios 1x1, 1x2, 2x1 and 2x2. For direct evaluation of the feature, it may require loading blocks of up to 6x6 pixels. It seems unnecessary to read data from the sliding windows (usually created as a register array with FIFO line buffers), and a better option is to read it directly from the addressable memory composed of BRAM. An optimal structure of this BRAM memory allows data reading without the use of a complicated multiplexer network. Zemčík and Žádník reduced the logic resources required for reading such large blocks by precalculating the subsampled versions of the image into memory. Reading only 3x3 blocks already precalculated in memory becomes sufficient for feature evaluation. The disadvantages are increased memory requirements and the need for storing differently sized subsampled versions of the original image. This has led to a complicated memory structure and high logic and memory resources demands.

Hereby proposed detectors are inspired by the detector introduced by Zemcik and Zadnik [44]. The main difference is that local image features of different sizes are not calculated from precalculated values, but directly from the original image. It saves memory resources, but on the other hand, it means reading blocks of different sizes (3x3, 3x6, 6x3 and 6x6 pixels), which is more complicated compared to the original constant size of 3x3. For simplicity, a 6x6 block (the worst case) can always be read and then subsampled as needed. A well-designed memory structure allows reading of unaligned 6x6 blocks at the same time, thus reducing logic and memory requirements.

Multiscale-detection

Designing an effective multi-scale detection on FPGA is an unresolved issue. Many related works do not address multi-scale detection at all [29, 31, 44, 43]. Several works [30, 32] suggest storing the entire image in BRAM memory, but this is not always feasible, especially at higher resolutions. Other works solve multi-scale detection by multiple image loading from external memory [34, 35]. Kyrkou et al. [33] reduced the number of loading from external memory by using more classifiers with different window sizes.

We have proposed a more efficient method of multi-scale detection. It does not require multiple image readings and uses significantly less memory than needed when storing entire images. The core is that the single scale detection requires only a narrow strip of the image memory, with the minimum height as the detection window. The same principle can be used for smaller versions of the image in multi-scale detection. Furthermore, it enables generating these smaller versions from the previous ones with fixed scale unit on-the-fly. The proposed method allows detecting objects of different sizes directly from the data coming from the sensor; the resulting system may not contain external memory at all, which would reduce the cost of the device. This approach can be further combined with the use of multiple classifiers with different window sizes. However, the memory requirements for storing multiple classifiers using LBP/LRD features are often greater than

the memory consumption when storing stripes of image. But the benefit always depends on the resolution of the input image, the height of detection window and the number of down-scaled versions.

Detector optimization

Parallel processing is one way to increase the performance of the detector on FPGA. Many authors [30, 29, 33] use parallel computation of more features in one position. Since the average number of evaluated features in one position is very small, it does not allow a high level of parallelization. When using the sliding window approach, it is necessary to evaluate all weak features and only then it is possible to move to the next one. In pipeline processing, premature rejection often results in a penalty meaning a speculative evaluation of other features or insertion of blank operations. Brousseau and Rose [35] suggested a method of evaluating features in neighbouring positions. The number of evaluated classifiers in specific positions is variable, which causes problematic divergence of the calculation. Besides, this approach combined with sliding windows leads to an increase of multiplexing network complexity, and thus an increase in logical resources. The detector introduced by Zemčík and Žádník [44], which is the basis for the proposed detectors, did not use any parallel processing.

We have proposed an approach for evaluating multiple positions in the pipeline simultaneously. This is possible by reading the data directly from BRAM memory, where the data for evaluating all positions of the entire line are accessed. It enables us to evaluate a bigger number of independent positions at the same time at various stages of evaluation. After evaluating the required features in one position, there is no need to wait for the evaluation of the surrounding positions; it is possible to move to the next position in the same line. This eliminates the issue with divergence and allows the creation of a longer pipeline without penalizing after the early rejection. The extension of the pipeline has a positive effect on the increase of the maximum circuit frequency and consequently, the rise of the detector performance.

We also suggest using more detectors connected in a cascade to increase performance. Each of these detectors performs detection in a different part of the image and at a different scale version. The optimal distribution can be precalculated so that the number of positions evaluated by each detector is approximately the same. This modification allows the detector's performance to be scaled very well for the needs of a specific application.

3.2 Goal of the thesis

The primary goal of this thesis is to improve the state-of-the-art in the field of object detection in the image on hardware platforms. The hypothesis is: *It is possible to design an object detector based on soft cascade deployed in programmable hardware with resulting precision comparable to the state-of-the-art, with real-time performance, with lower power consumption and less computing resource demands comparing to existing ones.*

The method of proof is creating a hardware detector that meets the required parameters and thus exceeds the state-of-the-art. Completing this task will require developing new methods and performing many experiments. In order to investigate of task the detector design, I have chosen to pursue the following methods:

- using local image features (LBP/LRD) and soft cascade classifiers in sequential engine with efficient block reading of image values for weak classifier evaluation,

- creating a multi-scale on-the-fly detector for high-resolution image data processing (without the need for external memory),
- using parallelization of weak classifier calculations by processing multiple positions at the same time, both at the level of sequential engine and cascade connection of multiple detectors.

The above proposed methods are being examined with the aim to confirm the presented hypothesis. The experiments will be performed in the detection of faces, pedestrians and license plates. Comparisons with other authors will be made on the face detection task, which is usually presented on other papers. For a fair comparison of performance due to different resolutions, detection window size, detection stride, multi-scaling, etc. a conversion to the number of processed detection windows per clock cycle will be used.

3.3 Core contributions

This thesis contributes to the state-of-the-art in the field of object detection in the image on hardware platforms. Three papers validating the hypothesis were published. They represent the experimental proof and demonstrate that it is possible to create the detector with defined parameters. The papers show that the proposed hardware detector outperformed the state-of-the-art in several aspects:

- better detection performance among boosted classifier – multi-scale face detection on Full HD (1920×1080 pixels) video at 60 fps (for object size 21 pixels and more) versus 640×480 at 30 fps by Hiromoto et al. [32],
- better detection performance in processed detection windows per clock cycle among all hardware detectors – up to 2.33 versus only processing detector with 1.97 by Jin [42] of full detector with 0.95 by Zemčík and Žádník [44],
- better performance/resources ratio – in all resources: LUT, REG and BRAM; the graphical comparison is in paper [2],
- better accuracy in face detection on CMU dataset [11] – recall 97 % with 0.2 false positives per image (FPPI) versus recall 91 % with 0.2 FPPI presented by Hiromoto et al. [32] and Kyrkou et al. [33],
- comparable accuracy in licence plate detection – aligned licence plates recall 99 % with 0.2 FPPI and unconstrained recall 98 % with 0.2 FPPI on own dataset

The contributions that implement experimental proof of thesis were presented in the following papers:

- *High performance FPGA object detector: Hardware prototype*, FPL[1] 2013. Paper with introduce an architecture of an engine for high-performance multi-scale detection of objects in videos based on WaldBoost training algorithm. The key properties of the architecture include the processing of streamed data and low resource consumption. The engine is implemented in FPGA and that it can process 640×480pixel video streams at over 160 fps without the need of external memory.

[1]International Conference on Field Programmable Logic and Applications

- *Cascaded Stripe Memory Engines for Multi-Scale Object Detection in FPGA*, TCSVT[2] 2019. Evolution of the previous paper witch expands performance and usability. FPGA detector can process a stream of image data so that it stores a narrow stripe of the input image and its scaled versions and uses a detector unit which is efficiently pipelined across multiple image positions within the memory. We show how to process images with up to 4K resolution at high framerates using cascades engine. As a detector algorithm use boosted soft cascade with simple image features that require only pixel comparisons and look-up tables; therefore, they are well suitable for hardware implementation.

- *Unconstrained License Plate Detection in FPGA*, submitted to VEHITS[3]. This paper shows the practical use of the previous detector in traffic application on the task of detecting unconstrained License Plate. To detect and localize license plates is use multiple sliding window detectors based on simple image features, each tuned to a certain range of projections. On a large dataset is detection rate 98%.

Results presented in these papers proof the hypothesis of this thesis.

3.4 Other publications

I am a co-author of some other publications dealing with other areas of image processing. I focused mainly on the effective implementation of the algorithm on the FPGA and the modification of the algorithm for stream image processing on the fly, ie without the use of external memory. List of my other publications in chronological order:

- *Single-Loop Approach to 2-D Wavelet Lifting with JPEG 2000 Compatibility*, SBAC-PADW[4] 2015 [86]. In this paper is presented a novel approach to 2-D single-loop wavelet lifting with can be efficiently pipelined in hardware. A newly developed 2-D core of CDF 5/3 wavelet filter is presented that, using a new sequence of operations, simplify the design. Moreover, the proposed approach, that uses one pass for 2-D transform, directly produces final output and reduces significantly the need for storing intermediate results into memory.

- *High Dynamic Range Video Concepts, Technologies and Applications*, Real-Time HDR Video Processing and Compression Using an FPGA, 2016 [87]. The chapter in the book deals with hardware acceleration of HDR video acquisition and compression. Individual HDR images are obtained by composing several differently exposed images obtained with a standard camera. Description of HDR compression and its implementation on FPGA.

- *True HDR camera with bilateral filter based tone mapping*, SCCG[5] 2017 [88]. In paper is presented a real-time HDR processing system evaluated on a custom hardware camera platform. They are proposal modifications of the the State-of-the-arts algorithms enabling efficient implementation on FPGA platform and real-time performance. The main focus of the paper is on acceleration of Durand local tone-mapping operator involving real-time bilateral filter. The proposed solution is compared to the existing research results in terms of speed, resource consumption, and numerical accuracy.

[2]IEEE Transactions on Circuits and Systems for Video Technology
[3]International Conference on Vehicle Technology and Intelligent Transport Systems
[4]International Symposium on Computer Architecture and High Performance Computing Workshop
[5]Spring Conference on Computer Graphics

The publications presented above do not directly contribute to the scientific goal and hypothesis validation of the dissertation. However, technologically they add to the options of using object detection in images. In some applications, for example, in bad light conditions such as sharp backlight, it is advantageous to combine object detection with HDR image processing to improve accuracy. In addition, the platforms created in these publications were used for experimental work with object detection.

Chapter 4

High Performance Architecture for Object Detection in Streamed Video

Before 2013, when this article was published, several successful attempts to create an object detector in hardware were proposed. However, their parameters were insufficient, and the practical use of these detectors was limited. In this paper, we have proposed a practical object detector in hardware that allows detected objects on video with resolution 640×480 and 160 frame per second.

Main contribution is to use boosted soft cascades of classifiers with local image features as weak classifiers. The combination of the unique structure of the memory and the local features enabled the effective sequential evaluation of weak classifiers. Another advantage is the new detection method, which allows the detection of objects of different sizes on-the-fly, i.e. without reloading the image and extreme demands on FPGA memory resources. Proposed detector outperformed state-of-the-art in better detection performance and better performance/resources ratio.

The work was also selected for a presentation within the FPL Demo Night [89], where it had good reviews.

The work builds on the previous work of Pavel Zemčík [44]. My contribution in this paper was designing a memory structure for efficient reading of image values per block for effective evaluation of a weak classifier. Also, I suggested on-the-fly multi-scale detection and implemented a detector simulator in C language to verify detector properties. Finally, I implemented and tested the detector in the VHDL language.

High Performance Architecture for Object Detection in Streamed Video

ZEMČÍK Pavel, JURÁNEK Roman, MUSIL Martin, MUSIL Petr a HRADIŠ Michal. High Performance Architecture for Object Detection in Streamed Videos. In: Proceedings of FPL 2013. Porto: IEEE Circuits and Systems Society, 2013, s. 1-4. ISBN 978-1-4799-0004-6 [1]

Author participation: 30 %
Conference ranking: A2 (Qualis[1])

Abstract

Object detection is one of the key tasks in computer vision. It is computationally intensive and it is reasonable to accelerate it in hardware, and especially in programmable hardware. The possible benefit of the acceleration is reduction of the computational load of the host computer system, increase of the overall performance of the applications, and reduction of the power consumption. In this paper, we shortly review the WaldBoost-based object detection algorithm and introduce a novel architecture of engine for high performance multi-scale detection of objects in video. We implemented the engine in FPGA and we show that it can process 640 × 480 pixel video streams at over 160 fps without the need of external memory, and with only modest consumption of FPGA resources. We evaluate the design, compare it to state of the art designs, and discuss its features and limitations. We conclude with remarks for future work.

4.1 Introduction

Object detection is one of the key methods used by the contemporary image and video processing applications, such as security and surveillance, production control, quality inspection, and human-machine interaction. One of most widely used methods [11] uses classifier to evaluate every sub-window of an input image in order to determine whether the area contains target object or not. In case of objects occurring in multiple sizes, the detection should be performed in multiple scales of the images or multiple sizes of the classification window.

Several authors proposed hardware implementations of object detectors [30, 34, 90, 33, 29, 44]. In most cases, they use cascade of boosted classifiers proposed by Viola and Jones [11] but other approaches to the detection, such as neural networks [91], are also used. Many object detectors use the original detection cascade with Haar features [30, 33] that are not particularly suitable for hardware implementation. Moreover, most of the designs use relatively large memory structures to store the input image. These drawbacks are avoided in the presented design.

The key properties of our architecture are the following. We use very simple feature extractors – Local Binary Patterns (LBP) [92] and Local Rank Functions (LRF) [93]. We replaced the cascade of boosted classifiers with WaldBoost [13] classifier, which provides improved detection speed and accuracy. At the same time, the engine does not need external memory storage as it requires only a narrow image buffer stripe which fits in the on chip memory.

[1] http://www.conferenceranks.com/

The architecture can handle streamed input, process it, and add detection results into the video stream in real-time A single instance of the detection engine implemented in FPGA can process 640×480 pixel video stream faster than in real-time – at over 160 frames per second with the clock speed of 152 MHz while consuming around 0.5 W of power. The accuracy of the detection measured on the task of face detection reaches over 85 % with one false positive detection per image. When synthesized, it consumes only modest amount of FPGA resources and thus multiple instances of the detection engine can be implemented in a single FPGA chip in order to boost the performance, or to enable detection of multiple object types at the same time. Special focus was put on minimization of the expensive memory structures usage and on low energy demand.

The paper describes the architecture, its implementation in Xilinx Spartan 6 LX45T FPGA, evaluates its properties on the face detection task, and compares it to the state of the art detection architectures.

4.2 Related Work

The following sections briefly review object detection with classifiers, methods of feature extraction, and methods of implementation of the detectors in hardware platforms.

Object Detection with Classifiers

Object detection with classifiers is one of the fastest methods for robust object detection in images. The best known and most widely used detector training algorithm is Cascade of boosted classifiers proposed by Viola and Jones [11]. The detection *cascade* subdivides the classifier into several increasingly complex classifiers (called *stages*). After evaluation of a stage, a decision about the class of the input image is made. Early stages can reject majority of background samples and thus the computational complexity is kept quite low. Every stage is composed from very simple elementary classifiers called *weak hypotheses*. More advanced method, *Soft cascade* [12], learns a *sequential classifier* which makes decision about image class after evaluation of every weak hypothesis. In this work, we use WaldBoost algorithm [13], which produces soft cascade classifier represented by a sequence of weak hypotheses $h^{(t)} = (\phi^{(t)}, \alpha^{(t)}, \theta^{(t)})_{t=1}^{T}$ where T is the total number of weak hypotheses. Every hypothesis contains parameters of feature extraction $\phi^{(t)}$, a list of responses $\alpha^{(t)}$ and a threshold $\theta^{(t)}$. Response of t-th weak hypothesis $h^{(t)}$ on image sub-window X is obtained by indexing $\alpha^{(t)}$ using the response of the corresponding image feature $f(X, \phi^{(t)})$, see Section 4.2.

$$H^{(t)}(X) = \sum_{k=1}^{t} \alpha^{(k)}_{f(X,\phi^{(k)})} \tag{4.1}$$

$$S^{(t)}(X) = \begin{cases} 0 & H^{(t)}(X) < \theta^{(t)} \\ S^{(t+1)}(X) & \text{otherwise} \end{cases} \tag{4.2}$$

The response of the strong classifier $H^{(t)}$ in every step is accumulated (4.1), and tested against $\theta^{(t)}$ (4.2). When the sum falls below the threshold, the evaluation ends as the input is likely to be the background. If the evaluation reaches the last weak hypothesis, the final decision is positive and it is likely that the object has been detected.

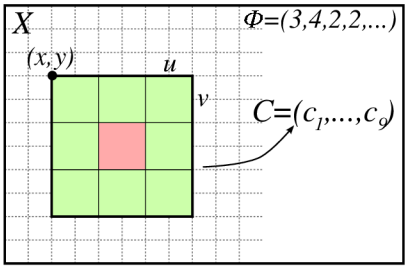

Figure 4.1: Nine cells C are taken from X according to feature parameters ϕ. Central cell c is marked by red, and border cells C^B are marked by green (applies for LBP evaluation).

Given an input image, the classifier is presented with its sub-images X from every position. Thus, every image area is classified and the positive final decisions are treated as detections. Multi-scale detection is usually solved either by *scaling of classifier window*, or *scaling of the image*. In this work, we explored both options.

Feature Extraction

So far, most widely used image features in both software and hardware are Haar features [94] as they proved to be a good information extraction tool for various tasks [34, 33, 29]. Haar features calculate differences of image intensities in adjacent rectangular areas. Other features, such as Local Binary Patterns (LBP) [92] or Local Rank Functions (LRF) [93], perform comparably or better than Haar features, and they offer interesting properties from the implementation point of view [93, 95, 44]. Another features, quite frequently exploited for detection, are Histograms of Oriented Gradients (HOG) [22], 3D Haar features [96], and others [97, 90]. Such features typically have good properties for various tasks, including detection of faces or pedestrians, but they are not suitable for hardware implementation.

In the presented work, we use LBP and LRD (subset of LRF) features, and parametrize them by their geometrical properties. LBP features are defined as $\phi^{LBP} = (x, y, u, v)$ where x, y corresponds to position in image X, and $u, v \in \{1, 2\}$ is size of feature cells in pixels. LRD features additionally contain identifiers of two selected cells a and b, thus $\phi^{LRD} = (x, y, u, v, a, b)$. A feature is evaluated from 3×3 grid of cells C represented by sums of their values in X, see Figure 4.1. Equations (4.3) and (4.4) evaluate LBP and LRD features respectively ($[\cdot]$ returns 1 when the comparison is true, 0 otherwise). Parameters X and ϕ impose C as mentioned above. LBP features compare central cell c with all other cells C^B. The results are concatenated to produce 8-bit word. LRD features calculate ranks of the cells a and b, and subtract them, producing results in range $\langle -8; 8 \rangle$. Rank is the number of positive comparisons of the cell value to all other cells.

$$f^{LBP}(X, \phi^{LBP}) = \sum_{i=1}^{8} 2^{i-1} \left[C_i^B > c \right] \tag{4.3}$$

$$f^{LRD}(X, \phi^{LRD}) = \sum_{i=1}^{9} \left[C_i > C_a \right] - \sum_{i=1}^{9} \left[C_i > C_b \right] \tag{4.4}$$

In both cases, the results of feature evaluation do not significantly depend on brightness and contrast in the image. Therefore, normalization of the image of any kind is not required.

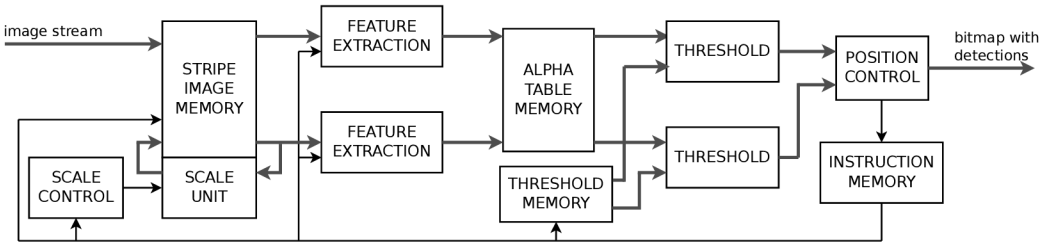

Figure 4.2: Block diagram of the proposed engine. The data for *Feature Extraction* blocks are loaded from *Stripe Image Memory*, weak hypotheses are evaluated using *Alpha Table Memory* blocks, and *Threshold* blocks evaluate strong classifier responses. *Instruction Memory* block holds the classifier program controlling the operation of the engine.

All the operations used in the feature evaluation are very simple (comparison, addition, subtraction) and thus the hardware implementation is relatively straightforward and it consumes only very little resources as shown in experiments.

Object Detection in Hardware

Since Viola and Jones published their real-time detection framework, much effort has been put into implementation of the detector in hardware, typically in FPGAs. Several designs for specific applications were introduced as well as general purpose detectors. Kim et al. [90] introduced real-time eye detector for FPGA based on AdaBoost classifier and MSC local image features. Cho et al. [30] proposed architecture implementing AdaBoost cascade detector with Haar features. In their approach, large memory is used to perform multi-scale detection on a pyramid of integral images. Huang and Vahid [34] also used Cascade classifier with Haar features and they tried to reduce resource requirements of integral image memory by a multiplex network and by constraints during the training process. A relatively low resources detection system was proposed by Zemčík and Žádník [44]. They used a WaldBoost classifier and LRD image features. They used only a small image strip to save resources of FPGA. Multi-scale detection was made possible only by an external DSP unit which precalculates an image pyramid. Kyrkou and Theocharides [33] introduced AdaBoost Cascade detector which combines two approaches to perform multi-scale detection – image downscaling and scaling of scanning window. Such design, however, does have relatively high consumption of resources as it uses classifiers with Haar features and normalization must be performed – in this case using square integral image. Moreover, usage of memory is high due to an extensive image buffering.

The proposed architecture is based on design proposed by Zemčík and Žádník [44]. Alike their architecture, LRD features and strip image memory are used. Proposed architecture brings a significant improvement in multi-scale object detection (no need of external memory resources), higher input image resolution, stream video processing and average FPS, all of this with only small FPGA resource footprint.

4.3 Proposed Architecture

The detector is designed as microprogrammed unit specialized in evaluation of weak hypotheses. Microprogram is synthesized from the results of machine learning process.

Representation of a Detector

Figure 4.2 shows block diagram of the engine. It works as a programmable automata driven by an instruction set with fixed size. Detector implements equations (4.1) and (4.2). It executes a classifier consisting of a long sequence of weak hypotheses on every image position and compares cumulative response to thresholds. A position is marked as positive when all weak hypotheses are evaluated. An instruction code for a stage t stores parameters for feature extraction $\phi^{(t)}$, stage identifier t for addressing table $\alpha^{(t)}$ and $\theta^{(t)}$, and additional information for engine control. The position of a feature is stored on 10 bits, size on 2 bits, ranks a and b for ϕ^{LRD} on 4 bits each, and identifier t on 10 bits. The whole instruction takes 64 bits. Items in α tables are quantized to 9 bits, and thresholds θ are stored on 16 bits.

The weak hypothesis evaluation block is pipelined to increase an overall performance. The pipeline has 9 stages, and therefore 9 weak hypotheses are processed in parallel. The instructions are loaded from the instruction memory and passed to the execution modules through delay units. The utilization of pipeline is almost 100 % given the ratio between the number of weak hypotheses to evaluate and clock cycles to fill the pipeline.

Feature Extraction Units

The detection engine implements LBP and LRD image features described in Section 4.2. The size of feature cell (u, v in ϕ) is limited to maximum 2 pixels and thus the features are limited to 6×6 pixel area. The position of a feature is not limited. Figure 4.3 shows four versions of cells that can be extracted form a feature area. The limitation to 6×6 pixels per feature does not adversely affect accuracy of the classifiers as shown earlier [93].

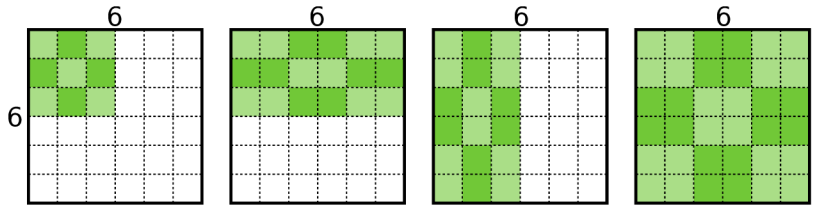

Figure 4.3: Cell configurations considered in this work. From 6×6 pixel area, four versions of 3×3 grid (shown in green) are extracted.

The block scheme of LRD feature evaluation, is shown in Figure 4.4. In the principle, DSP blocks extracts the grid of cells 3×3 cells. One of the versions is selected for evaluation according to feature parameters u, v. The ranks for a and b are calculated as a number of positive comparisons of selected cell values. The feature response is then calculated as the difference between the two ranks. Evaluation of LBP feature is similar – it is based on parallel comparison of central cell with the cells at the boundary.

The response of a weak hypothesis is finally obtained from the look-up table assigned to the feature using the result of the feature evaluation described above. The number of entries in the table for depends on the type of the feature. Typically it is 256 entries are used for LBP, and 17 for LRD (for implementation reasons 32 items are used).

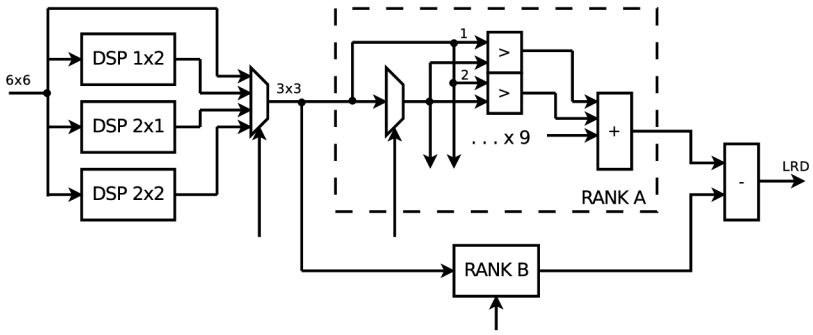

Figure 4.4: Circuit for LRD feature evaluation implementing (4.4). DSP blocks on the left extracts the four versions of cells. One versions is selected according to the current instruction. Two ranks are then calculated (circuit for calculation of one rank is shown in dashed block).

Memory Access Unit

One of the key requirements for the detection engine is the ability to process streamed data without excessive use of resources. Therefore, the proposed engine stores only a narrow image stripe holding a small part of the currently processed image and and its scaled versions. The scaled versions are calculated on the fly. The image strip is illustrated in Figure 4.5.

The memory is organized as a circular buffer of rows. In the proposed design, each 36-bit memory cell stores 6 image pixels (with 6-bit per pixel). For the purpose of the FPGA implementation, we convert 8-bit source image into 6-bit by discarding 2 least significant bits in order to save resources. This does not measurably reduce detection rate due to the properties of image features that are based on comparison of pixels only. Moreover, the image conversion can be reflected in the classifier training process.

Memory addressing is optimized for reading of 6×6 pixel blocks in every clock cycle. To enable this feature, interleaved addressing and 12 separately addressable memory structures must be used. This approach has significantly lower resource requirements than switching networks used for loading data from integral image for Haar feature evaluation. Also, the proposed solution outperforms the designs with external memory, DSPs, or large synthesized memory in FPGAs. In our experimental design, Xilinx Spartan 6 LX45T FPGA is used, and the memory is implemented in 12 BRAM blocks that store 32 lines of the image. The width of the buffer in pixels and the required number of the BRAMs used depends on the width of the input image (640 pixels in our case).

Image Scale Unit

The multi-scale detection is performed using scaled versions of the image (pyramid) in the image stripe stored in the image buffer. We explored two solutions that can be implemented in the engine.

Fine Image Scaling creates first octave of the pyramid using fine scaling unit (e.g. scale factor 5/6 with bilinear interpolation, which we use in this work). The subsequent octaves are calculated from the previous ones by downscaling with factor of 1/2. This process is illustrated in Figure 4.5. The performance of this solution is much higher compared to use of 5/6 scaling units only The approach results in slightly irregular scanning scales

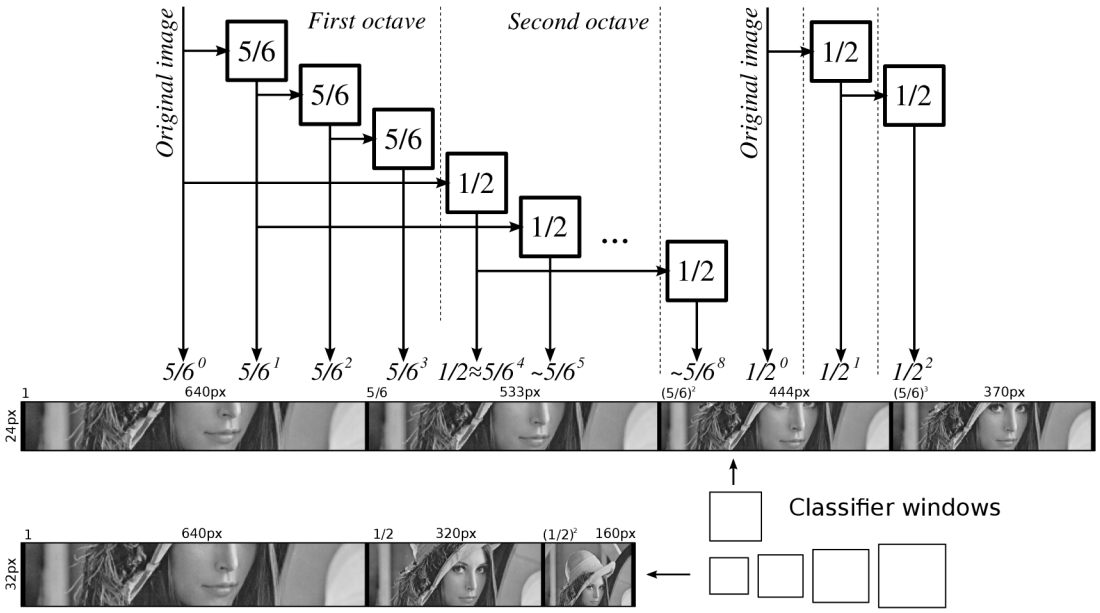

Figure 4.5: Illustration of image scaling approaches. Top: Fine scaling utilizing 5/6 and 1/2 scaling units, Middle: coarse scaling with only 1/2 scaling units, and Bottom: content of the image buffer stripe in the memory buffer for fine scaling option (top), and coarse scaling option (bottom). The original data are stored in the first 640 columns. Lower resolution scales are calculated on-the-fly from the original data.

as 1/2 is slightly different from $(5/6)^4$ but the difference is marginal (less than 2 %). The 5/6 scaling factor can be alternatively changed e.g. to 4/5 in case it is suitable for target application. A drawback of this approach is still relatively large memory requirement for complete image pyramid, and consumption of resources for the scaling unit. Even when only a narrow stripe is stored, its width can easily reach thousands of pixels.

Coarse Image Scaling scales images in memory by the factor of 1/2 only, as illustrated in Figure 4.5, and instead of the fine 5/6 scaling to use several classifiers with different window size. To cover the same scales as in case of 5/6 scaling unit, four classifiers have to be used – 18×18, 21×21, 26×26 and 31×31 pixels In this case the engine has to store four different classifier descriptions. The advantages of this strategy include decrease of memory space required for the downscaled images and also the reduced resource consumption fine scaling units are no longer required. The disadvantages, on the other hand, include the need to store more classifier definitions and the need to evaluate the classifiers at more positions. This is caused by the fact that with the size of the classifier window, the scanning step can not be adjusted to match the fine scaling version (e.g. the step of 31×31 pixel window should be $(5/6)^{-3} = 1.728$ pixels). In the present design, the step for all classifiers 1 pixel. This significantly increases the number of positions to evaluate from $828,885$ using fine scaling and 24 pixel window to $1,379,524$ using coarse scaling and the four detectors.

In the experiments presented in Section 4.4, we refer to different versions of the engine as 5/6 or 1/2. The 5/6 means that the engine uses fine scaling option and only one classifier. The 1/2 means that the engine uses coarse scaling option and four classifiers.

Accumulators and Thresholding Unit

The pipeline contains an accumulator for every classifier position being processed (presently, two 9 stage pipelines and thus 18 accumulators are used, see also Section 4.4). After evaluation of a weak hypothesis, the corresponding accumulator is updated with the weak hypothesis response and compared with a threshold assigned to the weak hypothesis. According to the WaldBoost evaluation strategy, the position is rejected if the accumulator value is lower than the threshold. Otherwise, the subsequent weak hypothesis is scheduled for evaluation. If all the weak hypotheses of the classifier are processed with no reject decision, the corresponding position is assumed to contain the target object. The length of the classifier depends on the application and results of machine learning process. The identified locations present the output of the detection process. The locations can be sent out in the form of a bitmap or just a list of positions with positive detection results based on the desired application.

Estimation of Engine Throughput

The theoretical maximal throughput (frames per second) can be empirically estimated using (4.5) where f is the operating frequency, n_p is number of pipelines, n_f is the average number of weak hypotheses per window, P is the total number of positions to evaluate in the image (and its scaled version), and c_s and c_i are constants reflecting the number of cycles required for image scaling and image loading respectively.

$$F = \frac{f \cdot n_p}{n_f \cdot P + c_s + c_i} \tag{4.5}$$

The detector speed is not necessarily a constant as it reflects *average* case. It can locally change with irregularities in data. It is faster when no target object is present in image, and gets slower with the number of objects, as every detected object requires all weak hypotheses to be evaluated.

4.4 Experiments and Results

The detection architecture was experimentally synthesized in a relatively small Xilinx Spartan 6 LX45T FPGA and evaluated on the Xilinx SP605 evaluation board[2]. The whole system is illustrated in Figure 4.6. Ethernet camera CAMEA Modicam M621 provides a 640×480 pixels video input at 60 fps The board is connected to a host PC through the PCI Express endpoint module.

In the experimental design, two pipelines for weak hypothesis evaluation are used to parallelise the detection process and to increase the overall performance. Image data, instruction memory, and classifier definitions are shared between the pipelines while the other units are replicated (address logic, multiplex network applied to the output of image memory, feature evaluation block, response accumulators, and thresholding unit). The theoretical throughput of the complete engine is 2 weak hypotheses per clock cycle (1 per pipeline). One of the pipelines, however, shares BRAM port with the image scaling unit and thus the performance is slightly reduced – down to approximately 1.85 weak hypotheses per cycle if the 5/6 scaling unit is used.

[2]VHDL sources, classifiers, and additional experimental results can be downloaded form `http://medusa.fit.vutbr.cz/fpga-engine`

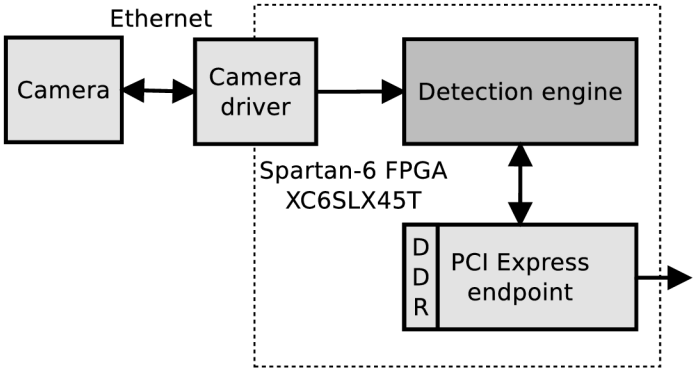

Figure 4.6: Block scheme of the architecture implemented in Xilinx SP605 Evaluation kit.

The performance of the engine was evaluated on face detection task. The detectors were trained by framework [62]. It supports all image features present in the detection engine, and it supports quantization of $\alpha^{(t)}$ and $\theta^{(t)}$ values. All classifiers were composed from 128 weak hypotheses. Although the classifiers used in the experiment contain only one type of features (either LBP or LRD), the detection engine supports combination of different types of features.

In the experiments, different engine versions were used to demonstrate how the different versions of scaling and image features affect the performance, resource consumption, and power consumption. The combinations were: LBP 5/6, LRD 5/6, LBP 1/2, and LRD 1/2 Each combination refers to a feature type used and scaling unit version. The 5/6 means that fine scaling units are used, and thus only one classifier is required for the detection. The 1/2 means that the image pyramid is created by 1/2 scaling unit, and thus four classifiers for detection must be used. In the versions with 5/6 scaling, the number of scales is limited to 7. This results in detection of objects in range of scales equal to approximately 4× magnification. In the versions with 1/2 scaling, the number of levels is in principle not limited but only 2 scales are created to match the 5/6 versions.

The results of the experiments can be subdivided into two basic parts – evaluation of properties of the classifiers and their detection performance in order to ensure feasibility of the architecture from the detection point of view, and evaluation of the resource consumption by the detection engine implementation in hardware.

Evaluation of Classifiers Properties

In the configuration with the 5/6 scaling unit, the detector uses a classifier with 24×24 pixels resolution. When the 1/2 scaling unit is used, four classifiers with different window sizes are used to emulate the 5/6 scaling step (18×18, 21×21, 26×26 and 31×31 pixels). Figure 4.7 summarizes classification performance of all the classifiers. The classifiers have similar detection rates regardless their size, and thus the use of different classifiers for different scales does not significantly change the accuracy of the whole system compared to the situation when only a single classifier is used. Figure 4.7 shows ROC curves of Viola and Jones Haar cascade (with 6,061 weak hypotheses) and our LRD and LBP detectors (128 weak hypotheses). The detection rates of our detectors are comparable to the detectors used by the state-of-the-art architectures.

The speed of the WaldBoost object detectors is determined by the average number of weak hypotheses evaluated per image position. Figure 4.7 shows average speeds of the

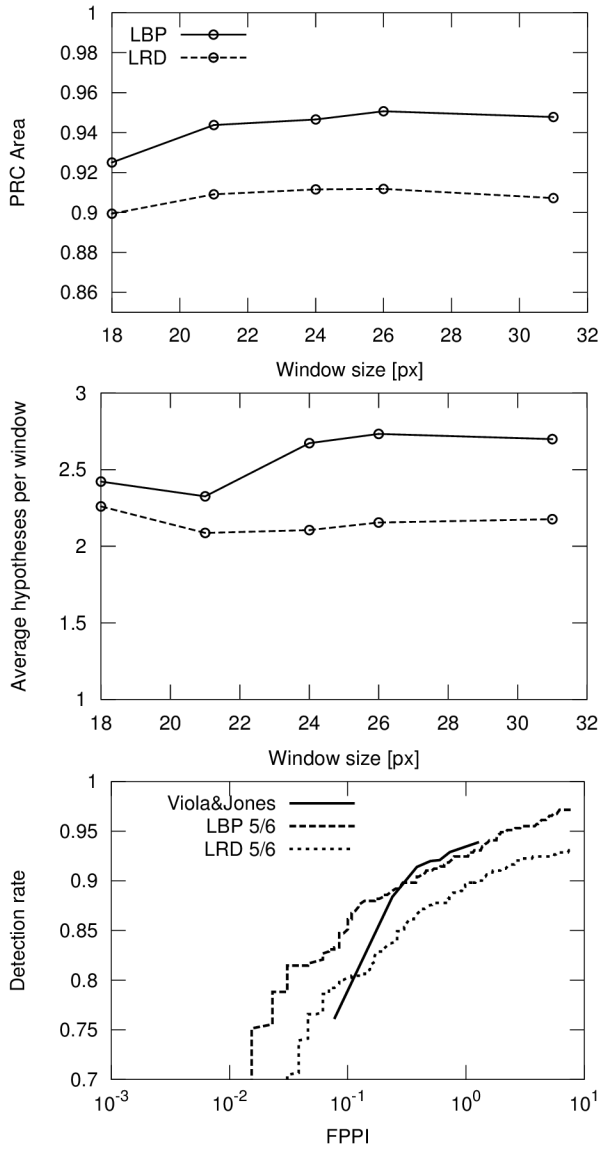

Figure 4.7: Evaluation of detectors on MIT+CMU dataset. Top: Area under Precision-Recall curve, Middle: Average number of weak hypotheses evaluated per window. Bottom: ROC curves of our detectors and comparison to Viola&Jones detector.

detectors used in experiments. Larger detectors tend to be a bit slower, but the relative difference between small and large detectors is not very significant to influence the overall speed of the engine.

In some cases, depending on the results of the machine learning process, the classification does not have to be done for each pixel position of the scanning window but e.g. for every second positions or every third positions without loosing the detection performance. If this is the case, the speed of the proposed engine is positively affected (in case of evaluation of every second position the speedup is approximately 4×) Of course, alternative forms of pre-processing to eliminate some image areas are also possible – e.g. color based processing, area of interest definition, or similar approaches. Such pre-processing is not present in the current design but it can be easily added.

Resource Consumption

Table 4.1 summarizes the resource consumption of the four different configurations of the presented engine and prediction of maximal throughput F according to (4.5). It shows that LBP versions use more BRAM blocks because, in general, LBP weak hypothesis prediction value tables are much larger than the tables in LRD weak hypotheses (256 in LBP and 17 in LRD). This fact can be observed especially in the versions with 1/2 scaling unit, which need more memory to store the four classifiers. Difference between the versions using the LRD features are marginal. Less demanding design versions are those with LRD classifiers, especially the LRD 1/2. Highest performance to resource consumption ratio is provided the LRD 5/6 variant as its performance reaches over 160 fps.

	FPGA Resources			Performance	F
	Registers	LUTs	BRAMs	[MHz]	[fps]
LBP 1/2	1678 (3%)	7098 (26%)	77 (66%)	163	91
LBP 5/6	1737 (3%)	7405 (27%)	43 (37%)	152	131
LRD 1/2	1673 (3%)	7014 (26%)	29 (25%)	163	107
LRD 5/6	1732 (3%)	7373 (27%)	31 (27%)	152	164

Table 4.1: Device utilization summary on Spartan6 LX45T (without camera and PCIe interface modules)

Results Discussion

The results of the experiments show that the proposed design is indeed capable of real-time detection of objects in video. We would like especially to highlight the ability to process streamed video without the need to use external memory. The design is scalable so several pipelines can be implemented in a single FPGA to boost the performance. Thanks to simple yet powerful image features the design consumes only very little resources and electrical energy while keeping competitive detection rate.

Still remain a few options how to modify the engine. Subsampling the image (e.g. by 1/2) before transfer to image buffer would decrease memory requirements, and increase the performance significantly. However such change would limit minimal size of detected objects to 2× size of the classifier. Classifiers with more weak hypotheses would increase detection accuracy. The number is limited only by on-chip memory. In such case, the throughput of the engine would be affected only slightly, as longer classifiers do not increase computational complexity too much.

Table 4.2 shows comparison to FPGA architectures for object detection that were proposed in recent years. Two variants of the presented design are listed – highest performance variant (LRD 5/6) and least resources (LRD 1/2). Our engines use less than one third of LUT tables and registers compared to detectors from [33, 29]. Compared to [30, 34, 90] it consumes only a small fraction of resources. Our engine requires more BRAM blocks than the other engines, but it does not need any external memory block or a DSP unit for the storage of images. Unlike the engines proposed by [33, 29, 44], our engine implements stream processing on a single FPGA chip.

The proposed design is suitable for applications that can benefit from embedded object detection, ranging from applications built into video cameras, where price and power consumption are critical, to applications where the computational performance needs to be offloaded to reduce the computational load of a host computer.

	FPS	Features	Scaling method	Scale factor	Freq. [MHz]	BRAMs	LUTs	Regs.	FPGA
Huang [34]	—	Haar	Img. scaling	1.2	65	—	80000	—	Virtex5 LX155T
Cho [30]	7	Haar	Img. scaling	1.2	—	41	66900	21900	Virtex5 LX110T
Kim [90]	50	MCT	Img. scaling	—	106	18	133000	45700	Virtex5 LX330T
Kyrkou [33]	40	Haar	Img./feature scaling	1.33	100	24	25800	23800	Virtex2 XC2VP30
Lai [29]	143	Haar	Img. scaling	1.25	126	44	20900	7800	Virtex2 XC2VP30
Zemcik [44]	22	LRD	None	None	—	14	2980	—	Virtex2 250
Our LRD 5/6	164	LRD	Img. scaling	1.2	152	31	7373	1732	Spartan6 LX45T
Our LRD 1/2	103	LRD	Img. scaling + multiple classifiers	1.2	163	29	7014	1673	Spartan6 LX45T

Table 4.2: Comparison of the presented engine with similar designs. The FPS column shows performance on 640×480 pixel input.

4.5 Conclusion

This paper presented a novel architecture for object detection in images and video using a scanning window and classification of its contents by a WaldBoost classifier. The main achievements of the new architecture include very high performance, multi-resolution detection, extremely compact design with no need of external memory, and generally low consumption of hardware resources.

Such design is possible thanks to the novel approach to feature extraction – features are not scaled but they are large enough to emulate smaller scaled features. This leads into a very small design of the detection engine. Multi-scale detection is achieved either by fine scaling of the image and use of fixed size classifier, or by coarse scaling of the image and use of four different classifiers. The platform is configurable and individual requirements can be reflected in selection of LBP or LRD image features and different image scaling strategies. The presented configurations range from the smallest design using LRD features with no scaling, to the fastest design – LBP with scaled image. The design is capable of processing 640×480 pixel video stream at over 160 fps with only a small consumption of resources, and it could be easily extended to process larger video frames.

Future work include improvements in the multi-resolution design, further reduction of resource consumption, improved implementation of multiple classifiers in one engine, and algorithmic improvements, such as prediction of neighborhood results.

Chapter 5

Cascaded Stripe Memory Engines for Multi-Scale Object Detection in FPGA

This article builds on the previous one and brings improvement of the parameters and usability of the detector. The results published in this article are used to verify the hypothesis. There is a detailed comparison of performance, detection accuracy and consumed resources with other works. For a fair comparison with other works, the conversion to the number of processed detection windows per clock cycle was made.

The main contribution of the article is a further increase in detection performance using more detectors connected in a cascade. The unique feature of the proposed architecture is the cascading nature of detector blocks, where one block passes re-scaled image data to the subsequent block in the chain. The slowest element in the chain then limits the total speed. We can generate the optimal distribution of processed parts of the image so that all detectors in the cascade are loaded similarly. It is possible to set the size of the stripe memory of each detector in the cascade individually and thus save space in multi-scale detection. Experiments with different cascade configurations are presented on the tasks of the faces and license plates detection. The resulting detectors outperform the state-of-the-art in detection performance represented by the number of processed detection windows per clock cycle. We introduced the first multi-scale hardware detector capable of processing a 4K image. The proposed detectors have a better performance/resources ratio compared to the state-of-the-art.

My contribution in this paper was the proposal of the cascade connection of detectors. I modified the detector hardware implementation in VHDL and created a tool to generate the optimal distribution for the cascade of detectors. By conducting a set of experiments on various cascade detector configurations, the properties were tested.

Cascaded Stripe Memory Engines for Multi-Scale Object Detection in FPGA

MUSIL Petr, JURÁNEK Roman, MUSIL Martin and ZEMČÍK Pavel. Cascaded Stripe Memory Engines for Multi-Scale Object Detection in FPGA. IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 1, page. 267-280. ISSN 1051-8215.[2]

Author participation: 30 %
Journal ranking: WoS: IF 4.133, Scopus: Q1 0,983

Abstract

Object detection in embedded systems is important for many contemporary applications that involve vision and scene analysis. In this paper, we propose a novel architecture for object detection implemented in FPGA, based on the *Stripe Memory Engine* (SME), and point out shortcomings of existing architectures. SME processes a stream of image data so that it stores a narrow stripe of the input image and its scaled versions and uses a detector unit which is efficiently pipelined across multiple image positions within the SME. We show how to process images with up to 4K resolution at high framerates using cascades of SMEs. As a detector algorithm, the SMEs use boosted soft cascade with simple image features that require only pixel comparisons and look-up tables; therefore, they are well suitable for hardware implemenation. We describe the components of our architecture and compare it to several published works in several configurations. As an example, we implemented face detection and license plate detection applications that work with HD images (1280×720 pixels) running at over 60 frames per second on Xilinx Zynq platform. We analyzed their power consumption, evaluated the accuracy of our detectors, and compared them to *Haar Cascades* from OpenCV that are often used by other authors. We show that our detectors offer better accuracy as well as performance at lower power consumption.

5.1 Introduction

Object detection in embedded systems is an important task that many applications of computer vision and scene analysis benefit from. Industrial quality control systems address various markers, traffic monitoring uses detection of cars and license plates, biometric systems detect faces and facial features, driver assistance systems detect cars and pedestrians. The detection is especially important in applications that directly rely on it, such as recognition or tracking, and in these applications, the speed, accuracy, power consumption, and/or robustness of detection matters most. In this paper, we address object detection implemented in embedded hardware. We focus on boosted detectors which analyze subwindows of an input image by a classifier composed from weak classifiers based on simple image features such as Haar [98] or Local Binary Patterns (LBP) [1]. Multi-scale detection is solved by scaling and processing of the input image in multiple resolutions – image pyramid. Embedded object detectors are often implemented directly in software using libraries such as OpenCV [74]. While this approach is easy and straightforward, it often is quite slow as detection is computationally demanding task and embedded processors tend to be simpler and slower than desktop CPUs. Another approach is to implement a custom detection algorithm exploiting various acceleration resources of the target platform –

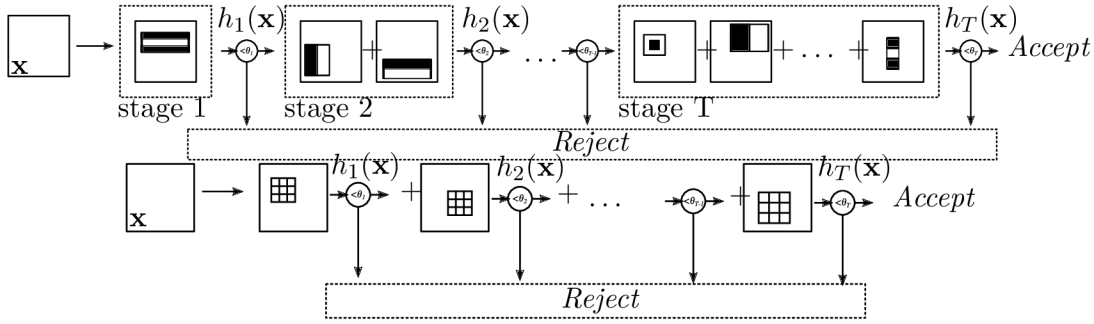

Figure 5.1: Comparison of Haar Cascade detector model (top) and Soft Cascade (bottom) that we use in our architecture. The main difference is that Soft Cascade does not contain *stages* and accumulates the response throughout the classifier. Another difference is that in Soft Cascade case the evaluation of the response can be terminated after every weak classifier.

CPU [95], GPU [99] or Field Programmable Gate Array (FPGA) [30, 1, 100, 68, 34, 33, 43] units. This is advantageous in many areas where the deployment of standard PC-based or embedded software solution is not possible, e.g. because of resource consumption, physical dimensions, industrial or military conditions, etc.

The object detection in embedded devices typically belongs to one of the three detection method categories. **1/** AdaBoost-based detectors – cascades of boosted classifiers [98] or soft cascades [57]. They typically use Haar image features [30, 31, 34, 33], or LBP [1]. **2/** Support Vector Machines (SVM) with Histograms of Oriented Gradient features (HOG) [22, 100, 65, 101, 68]; and **3/** Other methods implementing detection with background subtraction [52], keypoints [53], neural networks [102], or custom detection algorithms [54]. Most works, including thin one, belong to the first category, we give the detailed review of them in Section 5.3.

In this paper, we propose a simple and easy to use building block for FPGA that solves the object detection using state of the art boosted soft cascade classifier. We focused on implementation of the detection algorithm in the FPGA that efficiently utilizes the hardware resources and provides high performance. To produce classifiers for our hardware we used an existing, previously published algorithm [13]. The solution is multi-scale so it can detect objects of wide range of sizes.

It is suitable for various industrial applications, such as license plate detection, face detection, etc. The classifiers we use are especially suitable for hardware implementation since they are based only on pixel comparisons, look-up tables and integer-only calculations. Our architecture is extensively configurable, and it offers high image throughput even with high resolution inputs. In our main applications, which are detection of faces and license plates, we use processing of HD images (1280×720 pixels), but we also present a configurations for processing images with resolutions up to 4K (UHD, 3840×2160 pixels). IP Core for face detection and other resources are available online. Our contributions are specifically:

- Advanced memory architecture for image representation in block RAM (BRAM) which allows for simple and fast data random access suitable for fast feature extraction.

- Cascading of detector blocks which allows for increasing the input image resolution and the total performance.
- Multi-scale object detection directly in FPGA enabled by cascading od SMEs, without using external components
- Re-usable detector block that can be easily incorporated into other architectures using standard interfaces.
- Efficient streaming and pipelining and advanced control that fully utilizes the engine resources.

The paper is organized as follows. We start with a brief description of sliding window-based object detection in Section 5.2, where we introduce the framework common to many object detection methods. And we explain the difference between *cascade* and *soft cascade* classifiers. We continue with a review of existing works on object detection with boosted detectors in FPGA in Section 5.3. Section 5.4 contains analysis of existing solutions and describes improvements of the architecture proposed in this paper. In Section 5.5, we describe the soft cascade classifier model that we use in our architecture. We also briefly describe the classifier training algorithm. The proposed architecture is detailed in Section 5.6, where we describe the components of the detector. In Section 5.7, we compare the accuracy of our detectors to detectors from OpenCV, that are widely used by other authors, and compare our architecture to others works. Wa also analyze power consumntion of our architecture.

Finally, in Section 5.8, we present remarks on the performance of the presented architecture.

5.2 Detector model

Let us first describe a framework for object detection that sliding window-based methods have in common [98, 22, 13, 56]. We assume the input image $\mathbf{I}$ to be a grayscale raster and a classifier $H(\mathbf{x})$ a function that accepts or rejects the image patch $\mathbf{x}$ and returns a confidence estimation.

Detection on a single image

The detection function $D(\mathbf{I}, H, a)$ classifies every fixed-size patch of the input image $\mathbf{I}$ by the classifier H. A patch is defined by its location (m, n). Its size (u, v) is fixed, defined during classifier training stage. We use $\mathbf{x} = \mathbf{I}(m, n, u, v)$ for patch extraction from the location (m, n). The detection function (5.1) returns the set of locations accepted by the classifier and scaled by factor a, and the classification confidence.

$$D(\mathbf{I}, H, a) \in \{([m, n, u, v] \cdot a, H(\mathbf{x}))\} \tag{5.1}$$

Multi-scale detection

The detection process is illustrated in Figure 5.2. From the input image $\mathbf{I}$, a pyramidal structure $\mathcal{I}$ (see Equation (5.2)) with k scaled versions is created, such that $\mathbf{I}_j$ is $\mathbf{I}_{j-1}$ downscaled by factor $S < 1$. In our architecture, we use $S = \frac{5}{6}$, which results approximately in a pyramidal representation with 4 scales per octave.

$$\mathcal{I} = \{\mathbf{I}_0, \mathbf{I}_1, \dots \mathbf{I}_{k-1}\} \tag{5.2}$$

Figure 5.2: **(top)** The detection process on pyramidal image representation $\mathcal{I}$. The detection window of size u×v (yellow), is used for classification of every position m, n in image (example in red). The result of detection on each image is a set of locations $D(\mathbf{I}, s)$ accepted by the classifier (green). **(bottom)** The final detection result after non-maxima suppression.

The scale of j-th image in $\mathcal{I}$ can be retrieved as $s_j = S^j$; therefore, $\mathbf{I}_0$ corresponds to the original image. The result of the detection on $\mathcal{I}$, see Equation (5.3), is simply union of the results on individual images.

$$D(\mathcal{I}, H, S) = \bigcup_j D(\mathbf{I}_j, H, S^j) \tag{5.3}$$

The set $D(\mathcal{I}, H, S)$ is then processed by a non-maxima suppression (NMS) algorithm to suppress nearby detections and produce the final results for the image. We use a simple, overlap-based, NMS algorithm [56] which finds clusters of overlapping detections and keeps only the strongest detection from each cluster. However, other algorithms, such as mean-shift [22], could be used as well.

The main work in the detection process is done by the classifier H which scores the individual image windows. The classifier cascade introduced by Viola [98] (or sometimes called Haar cascade), is a widely used model, see Figure 5.1. The classifier cascade analyzes the input image patch $\mathbf{x}$ by a sequence of progressively more complex *stages* composed from *weak classifiers* based on simple image *features*. After evaluating of a stage, the image patch can be either rejected (classified as background) or passed to the subsequent stage. The soft cascade, shown in Figure 5.1, is not explicitly divided into stages and the rejection decision is made after evaluating each weak classifier. In this work we use the soft cascade

Table 5.1: Overview of state-of-the-art architectures. In *Length* column we report the number of features and number of stages for Cascades in brackets. *(self-organizing map neural network)

	Year	Image size [px]	Method	Feature type	Window size	Length	Scales	Object	Description
Lai [29]	2007	640×480	Cascade	Haar	20×20	52 (3)	15	Faces	Parallel calculation of feature responses
Zemcik [44]	2007	640×480	WaldBoost	LBP/LRD	31×31	80	-	Faces	Microprogrammable engine for feature extraction
Granat [31]	2007	256×256	AdaBoost	Haar	24×24	184	-	Faces	FPGA coprocessor for DSP
Cho [30]	2008	640×480	Cascade	Haar	20×20	2 135 (22)	18	Faces	Up to 3 features per clock cycle
Hiromoto [32]	2008	640×480	Cascade	Haar	24×24	2 913 (25)	18	Faces	Parallel calculation of feature responses
Martelli [68]	2011	640×480	SVM	Covariance	128×64	–	–	Peds.	Features extracted from 5x5 blocks
Kyrkou [33]	2011	320×240	Cascade	Haar	24×24	2 913 (25)	8	Faces	Combination of detector upscale and image downscale
Huang [34]	2011	320×240	Cascade	Haar	20×20	2 135 (22)	12	Faces	Scalable performance/resources tradeoff
Brouss [35]	2012	320×240	Cascade	Haar	20×20	2 135 (22)	15	Faces	Evaluation of multiple position in parallel
Jin [42]	2012	640×480	Cascade	LBP	20×20	250 (5)	16	Faces	Synthetic classifier
Kadlcek [43]	2013	1024×1024	WaldBoost	LBP	24×24	20	–	Faces	Synthetic classifier
Zemcik [1]	2013	640×480	WaldBoost	LBP/LRD	24×24	128	4	Faces	Programmable engine evaluating multiple positions
Yazawa [69]	2015	640×480	AdaBoost	HOG	40×80	–	5	Peds.,Cars	Featurtes extracted form blocks
Ma [23]	2015	1620×1200	SVM	HOG	64×128	3 708	34	Peds.	Featurtes extracted form blocks
Said [70]	2016	640×480	SVM	HOG	64×128	2 205	–	Peds.	Featurtes extracted form blocks
Kyrkou [65]	2016	800×640	SVM Cascade	LBP	20×20	1 062	18	Faces	Neural net pre-filter
Xu [103]	2016	320×240	Cascade	Haar	20×20	2 135 (22)	8	Faces	Features in FPGA, detector in ARM
Bilal [71]	2017	640×480	SVM	HOG	64×128	–	8	Peds.	Featurtes extracted form blocks
Yang [104]	2017	256×256	AdaBoost,SOM*	LBP	16×24	600	–	Faces	Heterogeneous parallel processor
Proposed	**2017**	**up to 4K**	**WaldBoost**	**LBP/LRD**	**up to N×27**	**1024**	**as required**	**Faces, LP**	Programmable engine with parallel processing of windows

model based on Local Binary Patterns (LBP) or Local Rank Differences (LRD) features and we describe this model in detail in Section 5.5.

5.3 Related work

Current cutting edge object detection algorithms are based on deep learning and convolutional neural networks (CNN). Generally, they achieve high detection accuracy in comparison to linear classifiers (such as Adaboost or SVM) [105, 106]. On the other hand, the computation of convolutional layers is very demanding; the number of operations required for evaluation is several orders of magnitude higher compared to linear classifiers. Furthermore, the neural networks usually require large amount of intermediate results, increasing memory requirements during inference. Another issue is the number of network parameters which can easily reach many milions. Furthermore, the memory requirements of CNN-based detectors are prohibitive for FPGA implementation. Current state-of-the-art FPGA architectures is that why can process only small images [107] and they are very slow [108],or they must use clusters of very large and expensive FPGAs[109]. For these reasons, linear classifiers are still favorable for implementation in FPGAs and embedded devices in general especially when processing of large images is required.

Table 5.1 summarizes important works in the field of embedded object detection from last ten years. Here we analyze the approaches the authors used.

Lai et al. [29] proposed a parallel hardware architecture based on Haar cascades. They achieved a detection speed up to 143 frames per second (FPS) at VGA resolution. Due to high demands on FPGA resources they limited the cascade to only first three stages (52 features), which led to low detection accuracy. Their implementation is therefore suitable as a preprocessing unit rather than full object detector. Cho et al. [30] implemented a Haar cascade-based face detection algorithm. They implemented various versions with one or three parallel classifiers to accelerate the processing speed. The disadvantage is high memory demand to perform multiscale detection on a pyramid of integral images.

Huang and Vahid [34] developed a method to generate a Haar feature-based object detectors. They aimed at automatic generation of detectors with a required precision for

FPGAs of various sizes. This approach allowed to reduce resource requirements of integral image memory and hardware complexity against universal implementation of detector. Brousseau and Rose [35] improved Haar cascade-based detector in FPGA by preloading of neighboring pixels, allowing parallel evaluation of classifiers in adjacent scanning windows. They also proposed a very complex evaluation control mechanism, allowing to rearrange execution of classifiers to coalesce the memory accesses.

Zemcik et al. [1], proposed an approach based on WaldBoost detection algorithm with LBP or LRD features. This approach implements stripe memory with block readout and image scaling but it is limited by fixed performace and by small image resolution, if the multi-scale detection is required.

Several authors proposed detection engines based on massive parallel execution of large number of features, increasing overall performance at the expense of the resource consumption. Jin et al. [42] proposed a design of fully pipelined classifier for high-speed face detection with LBP cascades. The features in each stage are executed in parallel. Kadlcek and Fucik [43] proposed an automatic classifier synthesis for the FPGA. Their method generates a fast image preprocessing unit with LBP features, processing complete detection window per clock cycle. High expense of FPGA resources allows for implementation of only limited number of weak classifiers.

Most of the works implement AdaBoost Cascade of classifiers with Haar features for face detection [30, 29, 34]. But, for example Kyrkou [100] detected traffic signs and cars. Some authors solve pedestrian detection with SVM[69, 70, 23, 71].

5.4 Design choices

In this section, we analyze significant works from the point of efficient hardware implementation and we summarize the outlines for the design of our architecture.

When it comes to hardware implementation, Haar features are not a good choice for several reasons. Haar feature is evaluated as a convolution of image and a mask. Each feature in the detector can cover a different and potentially large number of pixels, which means many memory accesses. Without using an integral image, this cannot be implemented to run in constant time, which is an important feature for pipelining in hardware. Using of integral image increases memory requirements as each pixel requires higher bit depth [30, 29, 34, 100]. When using integral image, each feature can be evaluated by referencing from 6 to 9 pixels, depending on the shape of the feature [30]. Reading these values from BRAM, unfortunately, means non-uniform memory access which cannot be executed in a single clock cycle; therefore, most of the works implement the sliding window as a register array with FIFO line buffers stored in BRAM. This allows for parallel access of pixels in the window and evaluation of multiple features in parallel. However, this also leads into a huge multiplexer network (20×20 search window requires 400:1 multiplexer [30, 29]), that occupies many resources in FPGA. The resource consumption increases dramatically with the size of the detection window and thus such architectures are constrained to use only small and fixed window sizes to save resources. Huang [34] solves this drawback by limiting feature positions and simplifying the multiplexers.

Zemcik [1] substitutes Haar features with LBP which replaces shift registers and delay lines by a set of BRAM memory blocks with organization that allowed for the multiplexing to be replaced by a simple block addressing technique. This approach has another advantage in pipelining of feature evaluation. It allows simultaneous processing of multiple image windows in the stream and thus full utilization of the pipeline, which is not possible with

standard scanning window approach [34, 31]. In general, the hardware detectors based on LBP features[42, 43, 65] achieves higher performance than Haar feature based detectors which is summarized in section 5.7.

Multi-scale detection is, in most cases, solved by storing the input image in RAM and scaling by an algorithm or circuitry independent on the detection unit [34]. Downscaled images are then passed to the detector from RAM one after another. Brouss [35] uses resolution so small that the image fits BRAMs in the FPGA. Kyrkou [100] combines image downscaling to half resolution and upscaling the detector window. Scaled version of the image is stored in BRAM. Granat et al. [31] scales the image features in the classifier and addresses the integral image at its original scale. Zemcik [1] scales image on the fly and stores only a narrow image stripe in BRAM. Some works [68, 43, 70] do not solve multi-scale detection and detects objects of a fixed size; therefore, their architectures are more simple and exhibits apparently higher performance.

As a basic building block in our architecture, we use an improved architecture by Zemcik et al. [1]. Specifically we improved the performance of pipelining, image scaling algorithm, the bit depth of the image and we extended it with cascading capabilites, described below. Our architecture differs from the others in several aspects. We use *soft cascade* instead of cascade of classifiers(see in Section 5.2). Soft cascade is usually more efficient in terms of the number of extracted features [56]. We use features that do not need integral image and that can be evaluated directly from the input image – LBP and LRD [62].

In our approach, the sliding window is not stored in FPGA registers. Instead, *Stripe Memory Engine* (SME) is used to store a narrow stripe of the input image in BRAM, see in 5.6. The stripe must be higher than the of classifier window (we use classifiers with height up to 24 pixels and stripe height is 32 pixels). In the classifier window, we limit geometric size of the features to 6×6 pixels which allows uniform reading of a fixed size pixel blocks from SME in one clock cycle. Juranek et al.[64] shows that limitation of feature block size does not have adverse effects on detector accuracy. Image is represented on 8 bits per pixel which saves resources compared to integral image where even 20 and more bits per pixel need to be used [100, 29]. The detector size is limited only by the height of the detection window but not by the width, which can be of virtually any size. We also do not use RAM to store the input image; instead, the image is scanned as it comes from the source and its scaled versions are generated on the fly (see in 5.6). Many image scales are stored in the same SME. The stripe memory, due to its organization, allows the evaluation of multiple scanning windows simultaneously and enables efficient pipelining and scheduling of the detector evaluation process. Moreover, dual-port BRAM allows us to implement two pipelines and therefore up to two features can be extracted in a clock cycle.

Another contribution of this work is that our detection engine can be cascaded in order to increase the performance and the image resolution using *Stripe Memory Cascades* (see in 5.6). It is basically a chain of SMEs where one SME sends the image data to the subsequent one. The number of instances is only limited by available resources. In practical setup, one instance can hold few high-resolution image scales and the other the rest low-resolution scales; therefore, the maximum image resolution is bigger compared to one instance solution. Moreover, both instances run in parallel and thersefore the performance is also increased. The number of instances in the cascade is limited only by available resources.

All of these differences – detector based on simple image features, image representation in SME, cascading and efficient pipelining – contribute to low resource requirements and overall performance of the proposed architecture.

5.5 Classifier model

The main part of the detection is the evaluation of the classifier $H(\mathbf{x})$ on image patches. It consists of the feature extraction and the classifier response accumulation, which we describe in the following text.

Feature extraction

Given an image patch $\mathbf{x}$, a feature extraction is a function $y = f(\mathbf{x}, \pi), y \in \mathbb{N}$ which extracts a value from $\mathbf{x}$ based on the parameters π. As a feature extraction function, we use Local Binary Patterns (LBP) with

$$\pi = (x, y, w, h)$$

or Local Rank Differences (LRD) [62] with

$$\pi = (x, y, w, h, a, b)$$

where x,y,w,h define the feature position and the size in the patch $\mathbf{x}$ and a,b are indices of two distinct cells in the LRD case.

The feature response $f(\mathbf{x}, \pi)$ is evaluated from values of 3×3 cells whose positions and sizes are defined by the parameters π. The cell values $\mathbf{c} = C(\mathbf{x}, x, y, w, h)$ are obtained as a sum of pixel values in the respective cell. The two feature types we use, LBP and LRD, differ in how the values $\mathbf{c}$ are processed.

Local Binary Patterns (LBP)

In general, LBP is based on comparison of pixels from a circular neighborhood to the central pixel and generating binary code [110], forming the feature output. Extended versions attempt to reduce the number of possible output values by rotating the resulting bit pattern or by restriction of the number of 0-1 and 1-0 transitions in the code [111].

In this work, we use a simplistic variant of LBP which takes 3×3 cell values and generate 8 bit code form comparison of the central cell to the border cells. Mathematically, the calculation can be written as Equation (5.4) where $>$ operator compares all values of a vector to a scalar value, resulting in binary vector. Weights $\mathbf{w}$ correspond to powers of two

$$w = [1, 2, 4, 8, 0, 16, 32, 64, 128],$$

so the dot product effectively sets the bits in the result. The zero weight, $\mathbf{w}_5$, corresponds to the central cell $\mathbf{c}_5$ which is used as a basis for the comparison. The range of the resulting values of $\mathrm{lbp}(\mathbf{c})$ is $[0; 255]$.

$$\mathrm{lbp}(\mathbf{c}) = (\mathbf{c} > \mathbf{c}_5)\mathbf{w}^\top \tag{5.4}$$

Equation (5.5) shows how the feature value is calculated, given an image patch $\mathbf{x}$ and parameters π.

$$f(\mathbf{x}, \pi) = \mathrm{lbp}(C(\mathbf{x}, x, y, w, h)) \tag{5.5}$$

Local Rank Differences (LRD)

Features based on local ranks proved to be successful in object detection tasks [62]. LRD uses scheme similar to LBP – processing of 9 values in 3×3 cells. It calculates the ranks of two distinct cells and outputs their difference. Mathematically, the function can be

described as Equation (5.6), where a and b are indices of two distinct cells. The resulting value of the $\mathrm{lrd}(\mathbf{c}, a, b)$ values is in $[-8; +8]$ range.

$$\mathrm{lrd}(\mathbf{c}, a, b) = \sum \mathbf{c} > \mathbf{c}_a - \sum \mathbf{c} > \mathbf{c}_b \tag{5.6}$$

Equation (5.7) shows how the feature value is calculated, given an image patch $\mathbf{x}$ and parameters π.

$$f(\mathbf{x}, \pi) = \mathrm{lrd}(C(\mathbf{x}, x, y, w, h), a, b) \tag{5.7}$$

The classifier

A classifier H is represented as a sequence of T weak classification functions

$$h_i = (\pi_i, \theta_i, \mathbf{a}_i), \quad i \in 1, 2, \ldots T \tag{5.8}$$

where π are parameters for feature extraction, θ rejection threshold, and $\mathbf{a}$ look-up tables with response values. Given an image patch $\mathbf{x}$, the response of the classifier of length t, Equation (5.9), is a sum of predictions produced by the individual weak classifiers.

$$H_t(\mathbf{x}) = \sum_{i=1}^{t} \mathbf{a}_i(f_i(\mathbf{x}, \pi_i)) \tag{5.9}$$

The sample $\mathbf{x}$ can be rejected (classified as background) after evaluating $k < T$ weak classifiers when $H_k(\mathbf{x}) < \theta_k$. And it is classified as detected object only if all T weak classifiers were evaluated. $H_T(\mathbf{x})$ is then used as classification confidence. The evaluation is summarized in Algorithm 1.

Algorithm 1 Evaluation of classifier H on the sample $\mathbf{x}$.

```
1:  procedure H(x)
2:      h = 0
3:      for t ← 1, T do
4:          (x, y, w, h, a, b) = π_t          ▷ Decode parameters
5:          c = C(x, x, y, w, h)              ▷ Extract cells
6:          g = lrd(c, a, b)                  ▷ Or g = lbp(c)
7:          H = H + a_t(g)                    ▷ Accum. the confidence
8:          if H < θ_t then
9:              return ('reject', 0)          ▷ Reject x
10:     return ('accept', H)                  ▷ Accept x
```

The number of features evaluated on a sample is, therefore, not fixed as each image patch can be rejected by different number of weak classifiers. The number of weak classifiers varies depending on the image patch content. We can statistically evaluate the *average number* of weak classifiers required for classification of a patch – $\bar{t}$. The value can be viewed as computational effort required for classifier evaluation. It can be calculated on a dataset using (5.10) by counting the number of evaluated weak classifiers W and classified image patches P.

$$\bar{t} = \frac{W}{P} \tag{5.10}$$

The value largely depends on the task and training data. Usual values values are $2 < \bar{t} < 5$ [64]. Lower values means faster detectors. For illustration purposes, later in this paper, we use $\bar{t} = 2.5$ which is a realistic value e.g. for face detection [64]. The value is especially important since it directly influences the performance of the proposed architecture, see Section 5.6.

In practise, the classifier of length T with LBP features is represented by three matrices $\mathbf{F}$, $\mathbf{A}$ and $\mathbf{T}$, where F is $4 \times T$ matrix with feature extraction parameters $\pi_t = (x, y, w, h)$, $\mathbf{A}$ is $256 \times T$ matrix with lookup tables $\mathbf{a}_t$, and $\mathbf{T}$ is $1 \times T$ matrix with rejection thresholds θ_t for each weak classifier. The t-th column of the matrices correspond to parameters h_t. Note that in the case of LRD features, the size of $\mathbf{F}$ is $6 \times T$ and the size of $\mathbf{A}$ is $17 \times T$, since LRD has six parameters $\pi_t = (x, y, w, h, a, b)$ and 17 output values for indexing. In Section 5.6 we use matrix $\mathbf{F}$ as a part micro program of the detection engine $\mathbf{A}$ and $\mathbf{T}$ are stored as lookup tables.

Classifier training

Detectors in this work are trained by WaldBoost algorithm [13]. But other algorithm producing a sequence of feature parameters and associates them with the corresponding response values can be used as well, e.g. [57]. The detailed description of the training algorithm is out of the scope of this paper since we focus mainly on the hardware implementation of the detection process. We kindly refer reader to the original paper [13]. Here we only provide informal description of the algorithm for reader to understand how it works.

The input of the algorithm is a pool of feature parameters, target *false negative rate* α, and a large set of training instances. E.g., when training a face detector, the training instances are image patches representing faces. The parameter α represents tradeoff between the final detector speed and its accuracy. Higher values of α (e.g. $\alpha = 0.2$) produces fast detectors with low value of $\bar{t}$, since they can reject background samples more rapidly. Low values (e.g. $\alpha = 0.01$) produces slower detectors with higher $\bar{t}$. We analyze this tradeoff in Section 5.7 on the task of face detection.

The training algorithm works in rounds, training weak classifiers one by one in s greedy manner. On the beginning of a round t, the algorithm loads background samples from a large set of images (not containing the target patterns) using the already trained classifier (i.e. weak classifiers from h_1 to h_{t-1}). For each feature in the pool, weak learner trains confidence values in lookup tables using AdaBoost [112]. In this step, the values are quantized to the resolution required by the FPGA. This is better than ex-post quantization (after the classifier is trained) since it allows training algorithm to adapt on errors caused by the computation with reduced precision [1, 44]. Then, the weak classifier minimizing exponential loss function [112, 98] is selected as h_t. Based on the distribution of H_t, for target and background samples, θ_t is selected so that as much as possible background samples may be rejected while discarding as few as possible target samples for the next round and satisfying target *false negative rate* α.

For the detector training in this work, we use our custom training software which produces detectors suitable for hardware, taking into account all possible quantization effects of the input image and values in lookup tables.

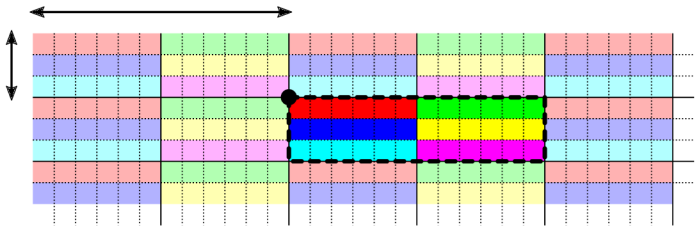

Figure 5.3: An example of SME in 2×3 organization and 6px blocks ($U = 2$, $V = 3$ and $B = 6$) stored in 6 BRAMS (color coded). Aligned block A of size 12×3 pixels can be retrieved from the memory in one clock cycle.

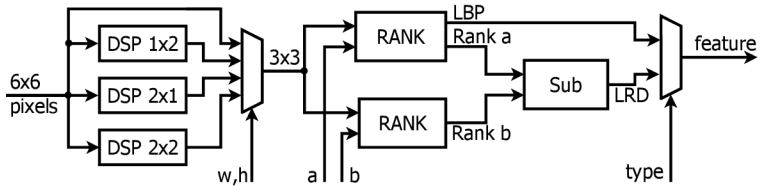

Figure 5.4: Circuit for feature extraction. The input is 6×6 pixel block from which, depending on feature parameters w, h, 3×3 cells are extracted. The resulting cells are used to calculate LRD or LBP features.

5.6 The architecture

We propose a hardware architecture that implements the key steps of sliding window object detection – image scaling, feature extraction, and classification of image patches. In the following text, we describe the design of the detector and its interface, and compare it to the equivalent software implementation in order to validate it. Figure 5.5 shows the overall schematic description of the detector.

Stripe Memory

The key part of our architecture is a *Stripe Memory Engine* (SME) which stores the active part of the input image and its scaled variants in multiple BRAMs, see Figure 5.5 for an illustration. When a new line is read from the image source, the data in SME are updated and scaled on the fly. The number of scales stored in SME is limited by the total width of SME raster, which is 4 096 pixels in this paper.

The architecture of SME is optimized for reading a block of pixels in a single clock cycle, so all data required for feature extraction are available in *constant* time. The data access is done in two stages. First, a fixed-size block aligned to certain position is retrieved from BRAMs to registers. Then, from this intermediate block, a sub-block with any size and alignment is retrieved by simple addressing. We store the image stripe in multiple BRAMs organized in a way that each BRAM is referenced only once when reading an aligned, fixed size block of pixels. BRAMs create a pattern of size U×V, and each BRAM stores B pixel block. This is illustrated in Figure 5.3 for U=2, V=3 and B=6. This requires $U \cdot V$ BRAMs to store the image stripe. Such an organization allows for reading $B \cdot U \times V$ pixel blocks (aligned to B pixels horizontally) in a single clock cycle by referencing all BRAMs.

Although SME can be configured almost arbitrarily, it is limited by the size of BRAM in the target platform. For practical applications, we use 4096×32 pixel raster, $U = 4$,

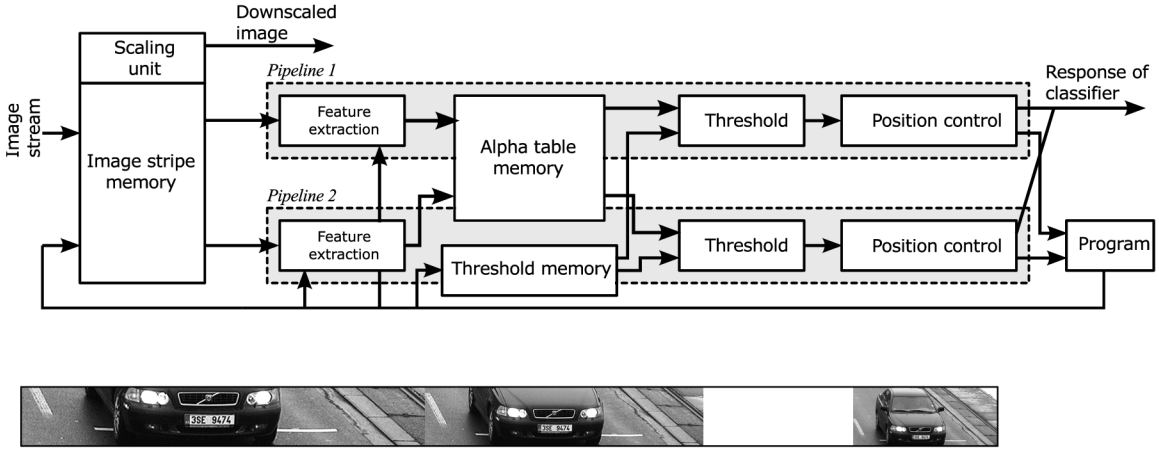

Figure 5.5: **(Top)** Block diagram of detector. The SME unit, which stores the image and produces downscaled images and two detection pipelines, driven by microcode program. **(Bottom)** Illustration of the stripe memory that we use for image representation and source of data for detector evaluation is shown, too. Incoming line (blue) is stored as a last line in the buffer. When possible, 6×6 blocks on the bottom of the buffer are scaled and stored as 5×5 blocks in subsequent scales. See the text and Figure 5.3 for details on how the data are stored in FPGA BRAMs.

$V = 8$, $B = 4$ and pixels represented on up to 9 bits. On our target FPGA, the SME takes 32 BRAMs with 36 kbit capacity. This organization allows us to retrieve 16×8 pixel blocks aligned to 4 pixel position. The, for feature extraction, we take 6×6 pixel sub-block or 8×8 sub-block for image scaling.

Feature extraction unit

The detection engine implements LBP and LRD image features. The size of the feature cells is limited to $w \leq 2, h \leq 2$, and thus the feature area is limited to a maximum 6×6 pixels. The position x, y is not limited in any way.

The block diagram of feature extractor is shown in Figure 5.4. The input is the 6×6 pixel block read from SME according to the absolute feature position in image (taking into account position of analyzed window). DSP blocks extracts all possible variants of $\mathbf{c}$ from the SME. One of the variants is selected for evaluation according to the feature parameters w, h from π. The ranks of elements a and b are calculated as the number of positive comparisons of an element $\mathbf{c}_a$ (resp. $\mathbf{c}_b$) to all other elements in $\mathbf{c}$. The ranks are subtracted to obtain the LRD feature value. Evaluation of an LBP feature is similar – parallel comparison of the central element $\mathbf{c}_{\mathbf{1},\mathbf{1}}$ to the elements at the boundary. The response of a weak classifier is obtained from the look-up table $\mathbf{a}$ associated with the extracted feature.

It should be noted that the circuit is designed to extract both LRD and LBP features simultaneously; however, in case that only one feature type is used, the circuitry for the other type is optimized-out during synthesis.

Detector control

The detector implements Algorithm 1. For every position (m, n) in SME image a sequence of instructions is executed. Each instruction reads the 6×6 pixel block from SME, extracts 3×3 cells $\mathbf{c}$, evaluates the feature function and accumulates the response value read from table $\mathbf{A}$. Then, the window is rejected or passed to the next stage based on the threshold value from $\mathbf{T}$. Everything is driven by the parameters from the instruction code. In case of rejection, new position is scheduled for evaluation. When all the instructions are finished, the window coordinates and the confidence value are sent to output.

The detector itself is controlled by a programmable automaton driven by a 32-bit instruction set. An instruction encode parameters for feature extraction – particularly the feature parameters from matrix $\mathbf{F}$ and sequence number identifier t for addressing matrices $\mathbf{A}$ and $\mathbf{T}$. We use 8 bits to encode each coordinate of feature position (x, y), 2 bits for (w, h), and 4 bits for each from (a, b). Note, that values a and b are present in the instruction code even for LBP-based detectors where they are unused. In the matrix $\mathbf{A}$, we store the response values on 9 bits and the thresholds in $\mathbf{T}$ table on 18 bits.

The detector microcode contains a sequence of up to 1 024 instructions; it means the length of the classifier is $T \leq 1024$. The number of instructions can be decreased or even increased, having linear impact on the memory requirements. Current implementation requires 1 BRAM for storing instructions, 1 BRAM for thresholds and 5 BRAMs for $\mathbf{A}$ and $\mathbf{T}$ in LRD case (64 BRAMs in LBP case) 2 BRAMs are occupied by instructions for static execution scheduler (see 5.6).

The evaluation of the classifier is pipelined. The pipeline is 14 clock cycles long and thus up to 14 positions are evaluated simultaneously. Thanks to the memory architecture described above, the pipeline can be utilized to 100 % which is impossible to achieve by previous scanning window approaches [34, 31]. We use two-port BRAM in SME, so we use two pipelines to double the performance. However, a small portion of memory accesses from the second pipeline needs to be reserved for image scaling and for storing the incoming image lines and the down-scaled data back to the SME – we leave one out of every 4 clock cycles for the scaling unit to generate the scaled images, and therefore the overall performance is $\bar{p} = 1.75$ features extracted per clock cycle.

Image scaling

Besides the original image, SME stores scaled variants of the image. The scaling is done on-the-fly over few last image lines. We use a block-based approach for scaling with fixed factor $S = \frac{5}{6}$, where 6×6 pixels blocks from a base scale are transformed to 5×5 pixels blocks in the subsequent scale. We implemented the separable, integer version of Lanczos [113] scaling algorithm for 8 bit images.

The process of downscaling and detector execution on individual SME lines is statically scheduled and driven by the microcode stored in BRAM. The classifier operations are performed to every line but every scale has a different number of lines to process. Moreover, the scaling is a block operation which is performed every 6-th line. This can cause occasional bursts of detector executions. The static scheduling allows us to distribute the execution of detector and scaling to avoid this execution bursts and ensure regular processing of image stream.

The maximum height of scanning window is given by height of SME minus size of block produced by scaling unit, which is 27px (32-5px in our case). This height is sufficient for

many of detection tasks and also standard detectors uses similar dimensions – 21×21 or 24×24 pixels [98, 13].

Detector interface

From the outside view, the detector is a computational block with one input, one configuration interface and two outputs. The input reads a stream of incoming image data of the given resolution. The configuration interface itself is composed from the detector definition (instructions and associated look-up tables), input image size, and sizes of scaled versions. The first output is a stream of the image data from the smallest scale in the SME. This output is used as an input for another detector instance. The second output contains detection results – coordinates and scale of detected objects. For both image input and output, we use AXI Stream Video interface for configuration the AXI-Lite interface and AXI Stream for detection results. This interfaces simplify integration of the detector to applications.

Stripe memory cascades

A single detector block is limited by the width of SME (4 096 pixels in our case) and by the performance of feature extraction, which is $\bar{p} = 1.75$ features/Hz (i.e. 350 M features/s with 200 MHz clock). From the performance point of view, it is not efficient to build the detector with a wider window (buffer) to hold more image scales, because the limitation of feature extraction speed would still remain. Our design allows for a more efficient solution – cascaded connection of detector blocks which we call *Stripe Memory Cascade*, illustrated in Figure 5.6. In the cascade, one detector instance generates scaled version of image and passes it to the subsequent instance. No limitation exists on the number of instances, except for the resources available on the target platform. All instances operates in simulataneously, effectively increasing the speed of the feature extraction. Output streams from all the instances are simply merged to form the output of the cascade.

Table 5.2 shows several configurations of cascaded instances, their performance, and resources they require. The naming convention we use for the configurations encodes the resolution processed and the number of detector block instances, e.g. `VGA/1` is configuration for processing of VGA image with one detector block instance and it is similar to what was proposed by Zemcik et al. [1]. Versions `HD/2`, `HD/3` and `HD/4` are configurations for HD image with different performance and resource requirements due to different assignment of image scales to detector instances. Versions `FHD/4` and `UHD/7` are for Full HD and 4K images. The versions for LBP and LRD differs mainly in memory requirements because LBP requires more BRAMs for classifier definition as described earlier in this paper.

Speed analysis

The theoretical maximum throughput (in frames per second) for one instance of the detector unit can be estimated using Equation (5.11) where f is the operating frequency, $\bar{p}$ the number of features extracted in one clock cycle, $\bar{t}$ is the average number of weak classifiers evaluated per window, and P represents the number of positions to evaluate in the image and its scaled versions assigned to the detector. The numerator of Equation (5.11) represents the total number of features extracted by the detector, the denominator is the average number of features that must be extracted on an image. As explained in Section 5.6, in our architecture has $\bar{p} = 1.75$. The value of $\bar{t}$ is the property the particular classifier, the average number of features that needs to be extracted from the image in order to decide the

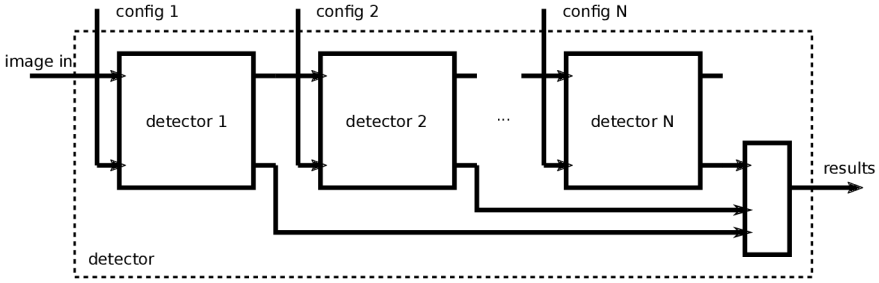

Figure 5.6: Cascading of detector instances. Each detector takes scaled input image from the previous one, the output coordinates and classifier response of detected objects are merged into one stream. Each detector is configured separately.

Table 5.2: Examples of cascade configurations, their predicted performances, and resource requirements. Valid for detector of size 21×21 px, $\bar{t} = 2.5$, and $f = 200\,\text{MHz}$.

Version	Feature	Res. [pixels]	Scales	Insts.	BRAM	REG	LUT	FPS
`VGA/1`	LRD	640×480	18	1	41	7640	9933	160
`HD/2`	LRD	1280×720	20	2	82	15292	19919	64
`HD/3`	LRD	1280×720	20	3	123	22944	29905	94
`HD/4`	LRD	1280×720	20	4	164	30596	39891	159
`FHD/4`	LRD	1920×1080	22	4	164	30596	39891	60
`UHD/7`	LRD	3840×2160	26	7	288	53552	69849	17
`VGA/1`	LBP	640×480	18	1	100	7650	9978	160
`HD/2`	LBP	1280×720	20	2	200	15312	20009	64
`HD/3`	LBP	1280×720	20	3	300	22974	30040	94
`HD/4`	LBP	1280×720	20	4	400	30636	40071	159
`FHD/4`	LBP	1920×1080	22	4	400	30636	40071	60
`UHD/7`	LBP	3840×2160	26	7	700	53622	70164	17

class of one analyzed window (see Section 5.5). It reflects the average case and it can change locally with irregularities in data that are hard to predict. We use $\bar{t} = 2.5$ for illustration purposes which is a realistic value for face detection. See Section 5.7 with the analysis of detectors we use.

$$F = \frac{f \cdot \bar{p}}{P \cdot \bar{t}} \tag{5.11}$$

The throughput of the whole cascade of detectors is limited by the slowest unit in the chain and it depends largely on sizes of images processed by the individual instances. In the Table 5.3, we show breakdown of `HD/*` variants from Table 5.2). Each version processes 20 image scales, the difference is in the manner how the scales are assigned to the detectors in the chain, and in the length of the chain.

Let us focus, for example, on the `HD/2` version, with two instances of detector. The first instance contains four image levels (resolutions from 1 280 pixels to 742 pixels of width) and we estimate around 5.4 M features need to be calculated on those four levels on average. Therefore, the speed of the first instance is around 64 FPS, calculated using Equation (5.11). The second instance contains the rest of the image scales (resolutions from 619 pixels to 42 pixels of width) and its speed is estimated to 233 FPS. The total speed of the `HD/2` is therefore 64 FPS as it is the minimal framerate from all detectors in the chain.

Table 5.3: Comparison of three different cascade designs, all for HD resolution. The values are shown for detector of size 21×21 px with average $\bar{t} = 2.5$ features per position, and $f = 200\,\text{MHz}$ clock. It can be observed that trade-off between the number of instances and desired detection performance exists.

Scale	Resolution	$P \cdot \bar{t}$	HD/2			HD/3			HD/4		
			$\sum P \cdot \bar{t}$	$\sum W$	FPS	$\sum P \cdot \bar{t}$	$\sum W$	FPS	$\sum P \cdot \bar{t}$	$\sum W$	FPS
1	1280×720	2200103	5468605	3979	64	3714188	2347	94	2200103	1280	159
2	1067×600	1514085							1514085	1067	231
3	890×500	1040628				3058318	2856	114	1754418	1632	199
4	742×417	713790									
5	619×348	488865	1497155	3536	233				1497155	3536	233
6	516×290	332887									
7	430×242	225972									
8	359×202	152945									
9	300×169	103230				193255	1312	1811			
10	250×141	68700									
11	209×118	45590									
...	...	...									
20	42×25	210									
FPS					**64**			**94**			**159**

Validation

During the design phase, we developed a software implementation of the detection algorithm which uses the same input data as the hardware implementation (look-up tables, instructions, thresholds, etc), and is based on the same image scaling algorithm. The architecture was validated by comparison of the results produced by the software implementation to the results produced by our architecture on a large set of images. The results were identical; therefore, we assume that the subtle differences in implementation in software and hardware are acceptable.

5.7 Results and Evaluation

In our applications we use Xilinx Zynq SoC with ARM CPU and FPGA. This combination allows for simple configuration of the detector and post processing of the results. However, if required, everything can be fixed and implemented in FPGA only. The design was written completely in VHDL with only few platform-dependent blocks (such as 36 kbit BRAM capacity); thus, it could be relatively easily adapted to various FPGAs, even from different vendor.

We built a prototype of a smart camera with HD CMOS image sensor and Zynq SoC Z-7020 chip. The camera captures image at 60 FPS and passes it through the `HD/2` detector. The detection results are processed on ARM core (non-maxima suppression, filtering) and the image along with the coordinates of detected objects are streamed through the network. We demonstrate the architecture on the detection of frontal faces and detection of license plates. As an example of our technology, we provide an IP Core of version `VGA/1` and `HD/2` detector with built-in face detector[1]. This IP Core takes approximately 15 % of Zynq Z-7020 resources.

[1] All resources can be downloaded from `https://github.com/RomanJuranek/zynq-detector`

Detector evaluation

Properties of WaldBoost detectors were experimentally evaluated many times on various problems [13, 114, 64, 115]. We tested our architecture in two example scenarios – face detection and license plate detection. These two applications are important in surveillance tasks. However, the detector can be used for detection of other rigid objects as well - cars [116], pedestrians [69] etc. We compare our detectors to the pre-trained detectors from OpenCV which implement Haar and LBP Cascades used by other state-of-the-art architectures. We report Receiver operating curve (ROC) – the tradeoff between *false positive rate* (the number of false detections generated per one image) and *miss rate* (the ratio of missed objects). Figure 5.9 shows a few images from each of the tasks.

Detection of frontal faces

We trained frontal face detectors on a large dataset of faces and compared them to OpenCV cascade detectors widely used by other authors as a baseline [30, 34, 35, 103]. The detector window size (u, v) was set to 24×24 pixels and the detector length to $T = 1024$. We trained four detectors with different target *false negative rate* $\alpha \in \{0.01, 0.05, 0.1, 0.2\}$, see Section 5.5 for details. From OpenCV, we used `haarcascade_frontalface_alt`, as it gives the best results from the built-in detectors. We tested the detectors on our set of 102 high resolution images with 1 857 annotated frontal faces (which is bigger and more challenging than MIT-CMU usually used for testing of frontal face detectors). The results in Figure 5.7 show that our detector (with $\alpha = 0.1$) gives almost 10× less false positives compared the detectors from OpenCV at the same recall level. The *recall* of OpenCV detectors is 94 % as reported by others [30, 1, 100, 68, 34, 33, 43]. Table 5.4 summarizes the speed and recall tradeoffs of the detectors trained with different value of α and their predicted performance in FPS when executed in version `HD/2` architecture. 60 FPS margin is satisfied by classifiers with $\alpha \leq 0.1$.

Detection of license plates

In law enforcement applications, such as speed measurement, detection of licence plates is a crucial step where accuracy and speed matters very much. We trained a license plate detector on a proprietary database of images taken by speed enforcement cameras. The dataset contains 30 000 automatically obtained samples of axis aligned license plates. The test set contains 1 000 images with manually corrected annotations. The dataset covers a wide range of conditions – day, night, sun, rain, snow and fog. For our experiments, the detector window size (u, v) was set to 84×12 pixels and the detector length was $T = 1024$. Accuracy evaluation in Figure 5.7 shows that the detection rate of WaldBoost detector is over 99 % when a false alarm occurs on one out of one hundred images. Detector speed measured on the test set is $\bar{t} = 2.7$, corrensponding to 62 FPS in `HD/2`. This is more than sufficient for this kind of application. For comparison, we trained Haar and LBP cascade from OpenCV on the same data using tools installed with the library. As Figure 5.7 suggests, our detector outperforms OpenCV detectors by a large margin.

Power Consumption Comparison

Table 5.5 shows estimation of power consumption of different platforms executing the face detection algorithm with Waldboost classifier ($\bar{t} = 2.5$, $\alpha = 0.1$) on 1280×720 images. On

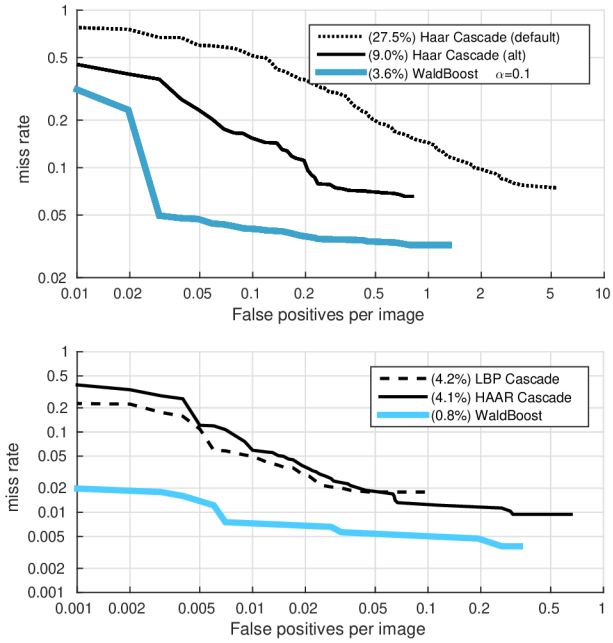

Figure 5.7: Accuracy evaluation of our detectors (WaldBoost) and comparison to OpenCV detectors (Haar and LBP cascade) for frontal face detection (top) and license plate detection (bottom). WaldBoost gives lower false positives at the same accuracy level.

Table 5.4: Speed analysis of our face detectors on `HD/2`. Fast detectors have slightly lower accuracy. The important value is $\bar{t}$, which directly influence the performance of our architecture.

	$\alpha = 0.01$	$\alpha = 0.05$	$\alpha = 0.1$	$\alpha = 0.2$
Recall	0.970	0.969	0.964	0.952
$\bar{t}$	7.54	4.43	2.50	1.96
F [FPS]	21	36	64	82

CPU, GPU, and Tegra, we used an OpenCL implementation of the detection algorithm from Herout et al. [99]. The critical steps were implemented in OpenCL and compiled for the target platform, efficiently exploiting SSE/AVX instructions and multi-thread execution on CPU and computing cores as well as texturing units on GPU. For the Intel CPU, we report maximum thermal design power (TDP). In case of GPU and SoC, the accurate chip power consumption is available. Power consumption of the FPGA was estimated using Xilinx Power Estimator, assuming the worst case with a 100% toggle rate (i.e. when signals flips every clock cycle). For the measurement purposes, we synthesized the `HD/2` in the different FPGA of the same family without the ARM core, so the results are not influenced by the power consumption of the ARM which is, in fact, not required during the detection. For all platforms, we report the metric which expresses energy consumed by the platform per one frame (Joules/Frame). The Table 5.5 shows that the FPGA design requires approximately five-times less energy than SoC Nvidia Tegra.

Table 5.5: Power Consumption comparison. FPGA-based detector is more power efficient compared to PC and GPU solutions.

	platform	FPS	Power (W)	mJ / Frame
PC	Intel i7 3770K	22	77	3500
GPU	GeForce 1080Ti	915	120	131
SOC	Tegra K1	38	4	105
HD/2	Artix7 xc7a75	64	1.52	24
HD/4	Artix7 xc7a200	159	2.95	19

Table 5.6: Comparison of critical parameters of the published architectures reported by the authors to the proposed architecture. Some works do not report all parameters or report *slices* (SL) or *logical elements* (LE). *Suitable only for preprocessing purposes

	Feature	Platform	Res. [pixels]	FPS	Stride	*wpc*	*f* [MHz]	BRAM	LUT	REG	DSP
Lai* [29]	Haar	Virtex2 VP30	640×480	143	1	0.886	126	44	20 900	7 800	–
Granat [31]	Haar	Virtex2 LX250	256×256	< 5	1	0.058	24	100	–	–	–
Cho [30]	Haar	Virtex5 LX110T	640×480	7	1	–	–	41	66 900	21 900	–
Hiromoto [32]	Haar	Virtex5 LX330	640×480	30	1	0.173	160	–	63 440	55 515	–
Martelli [68]	Covariance	Virtex6 LX240T	640×480	132	8	0.002	154	3	1 553 (SL)	–	22
Kyrkou [33]	Haar	Virtex2 VP30	320×240	64	1	0.083	100	24	25 800	23 800	–
Huang [34]	Haar	Virtex5 LX155T	320×240	100	1	0.268	65	–	80 000	–	–
Brouss [35]	Haar	Stratix4 GX530	320×240	50	1	0.083	125	–	–	–	–
Jin [42]	LBP	Virtex5 LX330	640×480	300	1	1.974	125	286	128 041	74 997	–
Kadlcek* [43]	LBP	Virtex2 LX250	1024×1024	130	1	0.948	137	17	1 007 (SL)	–	–
Zemcik [1]	LBP/LRD	Spartan6 LX45T	640×480	160	1	0.726	152	31	7 373	1 732	–
Yazawa [69]	HOG	Cyclone III	640×480	13	5	0.002	70	–	17 419 (LE)	11 306	–
Ma [23]	HOG	Virtex-6 LX760	1620×1200	10	4	0.045	150	381	46 238	186 531	190
Said [70]	HOG	Virtex6 LX240T	640×480	292	4	0.017	222	8	1 357 (SL)	–	46
Kyrkou [65]	LBP	Spartan6 LX150T	800×600	40	1	0.827	70	256	32 532	20 153	59
Bilal [71]	HOG	Cyclone IV	640×480	25	4	0.015	50	3	751	496	0
Ours LRD (HD/2)	**LRD**	**Zynq Z-7020**	**1280×720**	**64**	**1**	**0.930**	**200**	**82**	**19 919**	**15 292**	**0**
Ours LBP (HD/2)	**LBP**	**Zynq Z-7045**	**1280×720**	**64**	**1**	**0.930**	**200**	**200**	**20 009**	**15 312**	**0**
Ours LRD (UHD/7)	**LRD**	**Zynq Z-7045**	**3840×2160**	**17**	**1**	**2.334**	**200**	**288**	**69 849**	**53 552**	**0**

Comparison to other architectures

Table 5.1 shows the comparison to other architectures in terms of the maximum image resolution, detection algorithm and features, scanning window size, and type of detected object.

Due to our unique stream memory cascades, the detector can process images at very high resolution (up to 4K) while it is still capable of detection of very small objects. This property may be important e.g. when surveillance camera covers a large area. The most state-of-the-art architectures are capable of processing up to 1Mpix images.

The advantage of the proposed architecture, comparing to others, is the optional size of the detection window, which is limited only by the height of SME, while the width remains unlimited and freely adjustable. In other architectures [34, 31, 30, 35] the changing of window size means re-synthesizing of the whole design and, what is worse, larger windows takes more resources for multiplexer networks required for reading out the pixels. This is completely avoided in the proposed solution.

We compare our architecture to other similar ones. However, they are realized by different FPGA technology, have different input sizes, classifier strides in image and other parameters; these parameters are discussed in Section 5.8. To make comparison possible, we characterize all the architectures by number of processed scanning windows per clock cycle

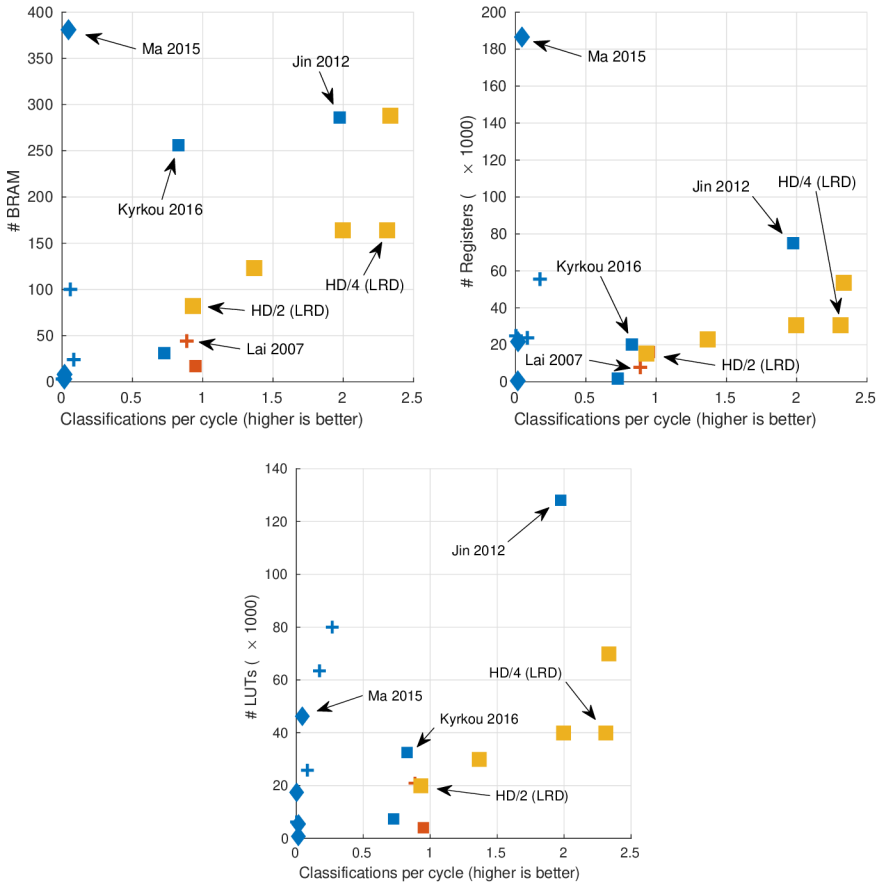

Figure 5.8: Comparison of classifications per cycle (*wpc*) and resource requirements of the architectures from Table 5.1. Yellow color encodes *proposed* architectures from this paper, red color *preprocessing units*, blue color the rest of the architectures. Marker shape encodes feature type used by the architecture (◆ stands for HOG, ✚ for Haar features, and ■ for LBP/LRD features).

(*wpc*) which gives raw performance measure independent on the used technology, frame rate and other parameters. Table 5.6 summarizes performance and resource requirements of our architecture and compares it to the other published works. Plots in Figure 5.8 show the dependency of resource consumption on the *wpc* classification for some architectures. Various proposed configurations of cascade instances (from Table 5.2) are plotted in each graph. It can be observed that overall performance increases with number of cascaded SME instances which proves its benefit.

The resource consumption and performance of configuration `HD/2` is comparable to Kadlcek [43] and Lai [29], anyway, their design achieves only low detection accuracy caused by very short detector and limits their possible usage for preprocessing purposes only. Our architecture, on the other hand, works as fully featured object detector while providing sufficient performance even at high image resolutions. In summary, the graphs on Figure 5.8 and Table 5.6 shows the performance superiority of LBP/LRD feature based detectors to Haar and HOG based detectors.

Figure 5.9: Examples of detected objects on selected images from testing datasets for face detection (top) and license plate detection (bottom).

The solution by Jin [42] and Kyrkou [65] achieves very high *wpc*, comparable to our `FHD/4` and `HD/2` configurations, but they require multiple-times more resources against our solution, even for only low image resolution. Comparing to work of Zemcik [1], the architecture we improved, we achieved higher performance, *wpc*, and maximum image resolution due to a cascading nature of our design.

SVM based classifiers using HOG features presented by Said [70] and Martelli [68] achieves high framerate mainly due to detection stride, where they process only every x-th image row and column, effectively making the image $x\times$ smaller, which reflects into low throughput. Moreover, they can only detect objects with fixed size 128×64 pixels as they do not solve multi-scale detection. This is why their architecture performs so well with so limited resources. However, to detect smaller objects, they need to upscale the image, and for multi scale detection, pyramidal representation need to be created, increasing the search space and slowing down the detection.

The proposed architecture achieves the highest performance (represented by *wpc*) compared to the others and also has a relatively low resource consumption as is evident from the Table 5.6 and graphs on the Figure 5.8.

5.8 Discusion

The unique feature of the proposed architecture is the cascading nature of detector blocks, where one block passes re-scaled image data to the subsequent block in the chain. The total speed is then limited by the slowest element in the chain. It is therefore possible to conveniently exchange resources for throughput.

In the architecture description, we were addressing the situation when all image windows and scales are processed – a full search. This is quite computationally demanding and, in many applications, unnecessary. The performance of the detection can be improved by several application-dependent methods. Scanning every n-th position in the image can increase the throughput proportionally to n. This can be used especially when detector window is large – e.g. in the license plate detection scenario above, we can scan every second or even third position often without serious impact on detection accuracy, while increasing throughput two or three times. In scenarios where the size of detected objects is known, or where we do not need to detect very small objects, image pre-scaling can be applied before it is passed to the detector. Moreover, the detector can be controlled to scan only certain scales. It can be controlled what lines are passed to the detection unit and overall performance can be improved by using only lines from certain image areas. This can be important e.g. in license plate detection where only license plates in the central part of the image are important for the application. During the development we always considered clock frequency $f = 200\,\mathrm{MHz}$, however, the frequency can be increased, especially when the design is simple and takes little resources, to improve the throughput. Another way how to increase clock frequency is to prolong the detector pipeline.

5.9 Conclusion

In this paper, we proposed and described in detail a novel architecture for object detection in FPGA. Our architecture uses detectors trained by WaldBoost algorithm with LBP or LRD image features, as opposed to AdaBoost Cascades used in other architectures. The key features of the architecture are simple representation of the detector by microcode, memory representation of the input image in *Stripe memory Engine*, cascading of detector units to scale performance, and parallel processing of image patches. We presented several configurations of detector cascades and analyzed their properties in detail. For research purposes, we provide an IP core for face detection in VGA and HD resolution. In the future, we will improve our architecture in order to include decision tree based detectors that promises better detection speed and accuracy.

Chapter 6

Unconstrained License Plate Detection in FPGA

This paper shows the use of the hardware detector in traffic application. The detector is a component of a smart camera which is used to detect unconstrained (wide variety of positions and angles) license plates for parking control in residential zones. The paper presented the first application of machine learning in hardware for license plate detection. The prevailing approaches used ad-hoc based methods, including morphological operations, segmentation and connected component analysis. Ït was not possible to compare the accuracy with other works because the test datasets are generally not available. Moreover, not many authors did unconstrained object detection, most of them operated with license plates in one rotation and size, for example, from images taken with a stationary camera above the road. The task of detecting unconstrained license plate is more complicated. Therefore, we performed a comparison with a standard detection method based on CNN and boosted classifier with ACF.

The results show that the proposed hardware detectors are suitable for real practice. They also allow us to build a cheap and compact camera with low power consumption. The technological goal of this dissertation thesis has been fulfilled by creating a functional and powerful hardware object detector with sufficient accuracy and low resource consumption.

My contribution in this paper was developing a modification of the previously published detector for processing several different classifiers. I trained selected standard CNN and ACF detectors to compare their accuracy with our detectors.

Unconstrained License Plate Detection in FPGA

MUSIL Petr, JURÁNEK Roman and ZEMČÍK Pavel. Unconstrained License Plate Detection in Hardware. International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS) 2021. Under review [3].

Author participation: 40 %

Abstract

In this paper, we propose an FPGA implementation of license plate detection (LPD) in images captured by arbitrarily placed cameras, vehicle-mounted cameras, or even handheld cameras. In such images, the license plates can appear in a wide variety of positions and angles. Thus we cannot rely on a-priori known geometric properties of the license plates as many contemporary applications do. Unlike the existing solutions targeted for DSP, FPGA or similar low power devices, we do not make any assumptions about license plate size and orientation in the image. We use multiple sliding window detectors based on simple image features, each tuned to a specific range of projections. On a dataset captured by a camera mounted on a vehicle, we show that detection rate is 98 % (and 98.7 % when combined with video tracking). We demonstrate that our FPGA implementation can process 1280×1024 pixel image at over 40 FPS with a minimum width of detected license plates approximately 100 pixels. The FPGA block is fully functional and it is intended to be used in a smart camera to parking control in residential zones.

6.1 Introduction

License plate detection (LPD) is an essential part of applications, such as detection of vehicles for traffic monitoring and enforcement purposes, or as a basis for automatic license plate recognition. In many industrial use cases, LPD is solved at a sufficiently advanced level in scenarios with restricted size, orientation, and in environment with controlled illumination (e.g. using IR flashes). A vast number of research papers has addressed this controlled scenario, and it is far beyond the scope of this paper to thoroughly review them.

In this paper, we focus on license plate detection implemented in FPGA in an image captured by *arbitrarily placed cameras*, *vehicle-mounted cameras*, or even *handheld cameras*. The target applications include parking control in residential zones, detection of stolen vehicles, and other applications, in which license plates of cars are automatically detected and recognised. Our goal was to improve the performance of license plate detection process, to implement the method in FPGA, and to provide the solution that requires low resources and that is suitable for integration into hardware devices, or even directly into cameras. The license plates captured in the image are severely deformed from their original rectangular shape by perspective projection, see Figure 6.1 for a few examples. Due to the character of the data, the detection and localisation can not rely on apriori information, such as license plate size or orientation in the image. Moreover, lighting conditions, ranging from deep shadows to overexposures, prevents using edge detection, gradient-based methods, and segmentation-based methods.

Figure 6.1: We detect license plates in wide variety of deformations in challenging light conditions. The images show the detected license plates with a confidence value.

An overview of several works on license plate detection with a focus on those implemented in embedded systems or PC is shown in Table 6.1. Many of the works are intended for stationary cameras and applications where the rough location and size of license plate is known [117, 118, 5, 119, 120, 121, 122, 123, 124, 125]. In many cases, they rely on the presence of the license plate edges to reduce search space.

Many methods [126, 119, 5] are based on connected component analysis (CCA) for the search of the candidate components in binarised images. [119] uses CCA with morphological operators and filters the components using knowledge about aspect ratio, size, and orientation. [126] segments characters within the CCA components by horizontal and vertical projections. [127] uses edge-based approach and rotation-free character recognition to search for tilted license plates. [128] uses CCA in combination with Radon transform to search for components with license plate properties (size, aspect ratio). [129] proposed license plate localisation on FPGA. They use two morphological operations and connected components labelling technique to find license plate candidates.

Template matching-based methods [118, 117] use background subtraction and edge detection for candidate search. Candidates are then scanned for characters using template matching. Other methods [120, 117] use template matching to find license plate candidates, and searches for characters in horizontal and vertical projections of the license plate images.

Machine learning based methods for licence plate detection typically use either SVM [121, 122, 123], cascade classifiers [124, 130] or Convolutional neural network(CNN)[131, 132, 133, 134]. [121] searches for candidate locations by horizontal and vertical edge detection and classifies them by SVM with co-occurrence matrix features. [122] presented a hybrid FPGA-PC approach where an image is filtered by chroma filter and gradient filter on FPGA, and then SVM is applied on candidate positions on PC. [123] also proposed adaptive thresholding, density filter and SVM with colour saliency features.

[130] proposes a Haar cascade detector on DSP for detection of cars and license plates. [124] uses cascade classifier with Local Binary Patterns (LBP) features and preprocesses the image by edge detection and morphology filters. [136] use AdaBoost cascade with LBP features for license plate detection and background subtraction for non-moving areas elimination. They use linear regression for estimation of size of detected plates. [131] use YOLO (You only look once) method [15] based on CNN to multi-directional license plate detection. [132] proposed a method combing the CNN with broad learning system based on AdaBoost for license plate recognition. [133] created CNN based encoder-decoder system where encoder detects candidate license plate characters and recognises them without considering the format of the license plate. [134] uses colour segmentation, corner detection and morphological operations to select the operating area, and finally uses a convolutional neural network to get license plate area.

Table 6.1: Summary of properties of state of the art detection algorithms. The accuracies reported by the works are not directly comparable as the methods were evaluated on different data and by different testing protocols.

	Rotation	Candidate search	Detection	Platform	Accuracy
[127]	tilt	Vertical gradient detection	Character recognition	PC	98 %
[122]		Chroma and gradient filtering	SVM	FPGA/PC	N/A
[119]		Edge detection, CCA	Filtering by size, orientation and aspect ratio	FPGA	99 %
[5]		Edge and background detection, CCA	Gradient statistics	PC	97 %
[121]		Edge detection	SVM	PC	88 %
[117]		Edge detection	Segmentation with hor. and ver. projections, template matching	FPGA	84 %
[118]		Edge and background detection	Character recognition with template matching	PC	84 %
[120]		Segmentation with edge detection	Filtering by size and aspect ratio	FPGA	99 %
[124]		Edge detection	LBP cascade	PC	94 %
[123]		Adaptive threshold and edge detection	SVM	PC	96 %
[131]	Free	–	CNN–YOLO	PC/GPU	99.5 %
[135]	Free	–	CNN	PC/GPU	98.35 %
[129]		Morphologic operation, CCA	Character segmentation	FPGA	–
[136]		Background subtraction	LBP cascade	PC	98 %
[132]	Free	–	CNN + AdaBoos cascade	PC/GPU	99.9 %
[133]	Free	–	CNN	PC/GPU	99.9 %
[134]		corner detection and morphological operations	CNN	PC/GPU	97.2 %
This work	**Free**	–	**soft cascade with LBP**	**FPGA**	**98 %**

Generic object detector hardware architectures have also been published, mostly demonstrated on face detection [30, 33, 1, 70, 2], that could be applied for detection of license plates. In most cases, they implement Haar Cascades [98] or SVM [22].

In this work, similarly to [130, 124], we propose statistical sliding window detectors trained by the machine learning algorithm. Such an approach is more robust than pure image processing, such as in [126, 119], since it easily adapts to visual variability of data. The difference in our approach from the other ones is that we use multiple detectors tuned to various deformations of license plates in order to boost the quality of detection. We use a large artificially generated dataset of training examples and propose an FPGA implementation of the detection process on Xilinx Zynq platform. Our architecture is based on [2] which provides excellent speed/resource tradeoff. We evaluated the method on a large image and video database obtained from industry.

To our knowledge, this work is the first to use boosted classifiers for unconstrained detection of license plates directly in FPGA. Other works that expoit FPGA use mostly segmentation or filtering approaches which are targeted for specific scenarios (and usually fail in unconstraned detection). Recent works employ mostly neural networks which are superior in accuracy of detection but their implementation in FPGA is limited and expensive [49, 51] or they require specific hardware (GPU, VPU) [131, 132, 133, 134].

6.2 License plate localization

Our task is to detect license plates observed by a camera and to localize them; see Figure 6.1 for example images. The intended applications require high accuracy of the detection result. At the same time, it can tolerate a reasonable amount of false detections (which can be filtered out later, for example, by the license plate recognition process). Our license plate detection algorithm is based on multiple independent boosted classifiers, whose results are merged to get the final results. Each of the classifiers captures license plates with a specific range of observed in-plane rotations. We use constant soft cascade classifiers [13, 57] with Local Binary Patterns (LBP) features [110, 1]. The detection process is illustrated in Figure 6.2,

Figure 6.2: **(left)** Image scales and objects detected by the classifiers. **(right)** Final detections after non-maximum suppression. The classifiers are color-coded, the yellow one detects license plates almost aligned, the green one detects slightly tilted license plates.

Table 6.2: Ranges of orientation ranges amd window sizes for the three classifeirs.

Range [°]	Window [px]	LP images
$0 < \phi < 15$	18×64	
$10 < \phi < 30$	22×56	
$25 < \phi < 45$	28×43	

The classifier

The classifier is a function $H(\mathbf{x})$ which gives the confidence value for an image patch $\mathbf{x}$, formally defined in Equation (6.1). During the detection process, every location of the input image is analyzed by the classifier. Multi-scale detection is solved by image scaling by a fixed factor. The classifier $H(\mathbf{x})$, is a sequence of T weak classification functions ($T = 512$ in our experiments), and its response on image window $\mathbf{x}$ is a sum of predictions produced by the individual weak classifiers. We use simple weak classifiers based on LBP features f and lookup tables A with the confidence predictions.

$$H_t(\mathbf{x}) = \sum_{i=1}^{t} A_i(f_i(\mathbf{x})) \tag{6.1}$$

The detection process on an image $\mathbf{I}$ produces a set of locations and sizes that were not rejected by the classifier, Equation (6.2). Each candidate comprises of its location x, y in the image, size w, h and confidence score c (the classifier response on the image patch corresponding to the location).

$$H(\mathbf{I}) = \{(x, y, w, h, c)\} \tag{6.2}$$

An important property of soft cascade classifiers is that the classification function $H(\mathbf{x})$ can be terminated after evaluating k-th weak classifier, when $H_k(\mathbf{x}) < \theta_k$. Thresholds θ are trained, so that majority of background samples is rejected early in the process. The

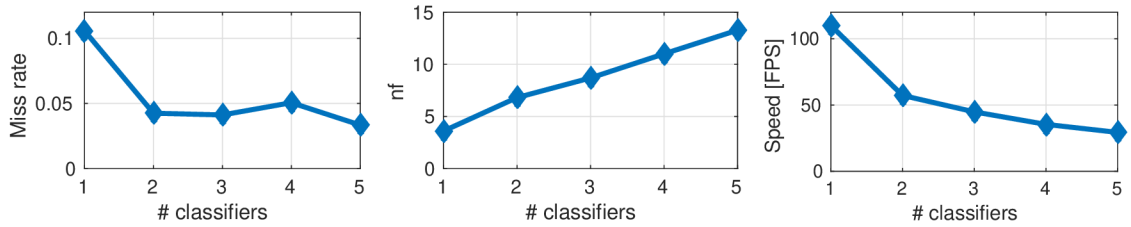

Figure 6.3: With the number of classifiers used by the detector, the average miss rate **(left)** reduces, and computational complexity **(middle)** increases linearly which reduces the performance of our implementation in FPGA **(right)**. We choose to use 3 classifiers.

computational complexity of the classifier dramatically decreases compared to the case of evaluation of all T weak classifiers, and it can be evaluated as an average number of weak classifiers evaluated per image position nf (which is usually orders of magnitude lower that T). This value is especially important as it directly influences the speed of the detector in the FPGA implementation.

Detection with multiple classifiers

We propose to use n classifiers $\mathcal{H} = \left\{H^{(1)}, \ldots, H^{(n)}\right\}$ for the localization of license plates, each tuned for LPs with specific range of rotation angle ϕ. Each classifier produces detections independently, and the final set of candidates is obtained as a union of all candidates (6.3). Final locations are produced by a simple, overlap-based non-maxima suppression algorithm.

$$\mathcal{H}(\mathbf{I}) = \bigcup_i H^{(i)}(\mathbf{I}) \tag{6.3}$$

Each detector is trained for a specific range of license plate angles ϕ. The range assigned to the k-th classifier is,

$$\max\left(\frac{45(k-1)}{n} - 5, 0)\right) < \phi < \frac{45k}{n},$$

where 45° marks the upper limit of license plate orientations. The classifier window aspect ratio is set as a mean aspect ratio of license plates falling in the range ϕ. The window size is set to constant area of 1024 pixels and 2 pixel margin is added. The ranges and window sizes for $n = 3$ classifiers are summarized in Table 6.2.

The number of classifiers in the ensemble $\mathcal{H}$ influences the accuracy and speed of the detection process. One classifier can not capture all the variability of the license plates, while more detectors are more accurate but slower. This trend is shown in Figure 6.3. Two classifiers give already reasonable detection rate and sufficient speed. We observed that using more classifiers results in better localization. Using more classifiers, however, reduces speed, and for this reason, we use three classifiers in our hardware implementation (Section 6.3).

The training data we used is a broad set of license plate images randomly transformed to match deformations likely to occur in the target application, since the scenario is often known and fixed, e.g. in case of our vehicle-mounted camera. Few samples are shown in Table 6.2. The advantage of such an approach is that it can be automated and any

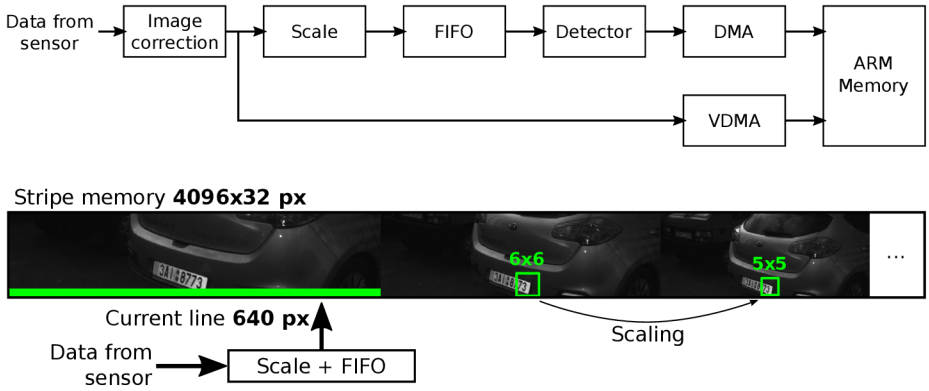

Figure 6.4: **(Top)** Block scheme of the camera prototype. **(Bottom)** The data from sensor are passed to a narrow image stripe in memory. All scales are created automatically.

number of training samples with precise ground truth (including orientation) can be quite effortlessly generated for each application case.

6.3 FPGA architecture

We implemented the LP detector on our hardware camera platform based on SoC Xilinx Zynq XC 7Z020. The SoC contains two ARM CortexA9 cores and FPGA interconnected through high-speed AMBA AXI bus. The approach is, however, quite general and can be applied to almost any Xilinx Zynq family member and also to other platforms. In our case, the platform is equipped with a low noise global shutter CMOS image sensor Python1300 from ON Semiconductor, which is attached directly to FPGA, forming a smart camera. The sensor resolution is 1280×1024 pixels and it can capture up to 210 FPS.

Figure 6.4 shows the block diagram of the camera with the detector integrated into it. As the minimum size of the license plate needed to detect is approximately 100 pixels in width, we downscale the input image to half resolution before the detection phase. The image is passed to the detector block through balancing FIFO that covers irregularities in the detector speed that depend on the image content and that are hard to predict. The results of detection – locations of detected objects – are transmitted to ARM CPU memory, along with the original image. On the CPU, the detected license plates are tracked using the Kalman filter. The camera outputs cropped license plate images that are transmitted to the server for further processing (OCR, storage, etc.).

The Detector block is based on [2] which implements multi-scale soft cascade detector with LBP [110] or LRD features [62]. The engine works as a programmable automaton, using a sequence of feature parameters as instructions. We modified the engine in order to use three classifiers and adapted it to a more recent platform with more resources and memory.

The engine works without external memory, storing only a narrow stripe of the image which is being stored and analyzed directly in BRAM inside the FPGA. The stripe memory size is 4096×32 pixels in 32 BRAMs organized in a way that data for a feature evaluation can be obtained in one clock cycle. The stripe memory is filled from the CMOS, and the scaled versions of the image are created on-the-fly; the process is illustrated in Figure 6.4.

The classification function is executed overall positions in the stripe memory. The processing is heavily pipelined, we use two pipelines of length 9, so up to 18 positions are

Table 6.3: FPGA resource utilization

	BRAM	LUT	REG
Image acquisition	5	2559	4746
Balancing FIFO	4	115	210
Detector	38	9521	7099
Storage	5	3734	4785
Total	**52**	**15938**	**16840**
7Z020 res.	**37 %**	**30 %**	**16 %**

processed in parallel. The efficiency of feature extraction is $np = 1.75$ features extracted in clock cycle. Overall performance in frames per second, expressed by equation (6.4), is, in general, dependent on the clock frequency f and the total number of features that must be calculated on an image for all classifiers (i.e. the number of positions P times the average number of features nf for each individual classifier).

$$F = \frac{f \cdot np}{\sum_{i=1}^{k} P_i \cdot nf_i} \tag{6.4}$$

In this work we assume $f = 200\text{MHz}$; $P_1 = 872487$, $P_2 = 899067$, and $P_3 = 915442$; and $nf_1 = 2.89$, $nf_2 = 3.23$, and $nf_3 = 2.78$. The values of nf were estimated on a testing set of 1522 images and are valid for the experimental classifiers presented in the Section 6.4. The estimated upper limit of performance of the detection unit is therefore $F = 48$ FPS. Table 6.3 summarizes resources required by the design. The whole solution takes approximately one third of resources of the 7Z020 FPGA which we use in our solution.

6.4 Detector evaluation

We evaluated the proposed method in two modes – *Single image* mode and *Tracking* mode.

In the *Single image* mode we evaluate precision-recall trade-off on a large number of testing images. However, it is impossible to compare to other published license plate detection methods (summarized in Table 6.1) since they published results on different testing datasets using different testing protocols. Moreover, none of them has available source code. For this reasons, we compare our method to well known general object detection methods – ACF [57] and MTCNN [137], and a commercial solution Plate Recognizer[1]. It must be however noted that none of them is targeted for FPGA with all of its constraints (low memory, integer arithmetics, etc.). MTCNN is a recent method based on neural networks and it require PC/GPU platform. Plate Recognizer internal structure was not published. ACF is a method similar to ours since it shares a common classifier structure and the detection algorithm. For the evaluation, we used our internal dataset of 1 522 images with 811 manually annotated license plate bounding boxes.

The results are summarized in Figure 6.5. Not surprisingly, MTCNN and the commercial solution give almost perfect results. ACF is comparable to our method in terms of recall but with higher precision. The results for our method show that a single classifier already gives reasonable results and adding more classifiers tuned to different transformations further improves them (see also Figure 6.3). Our method can reach recall over 95%

[1] `www.platerecognizer.com` provides free REST API

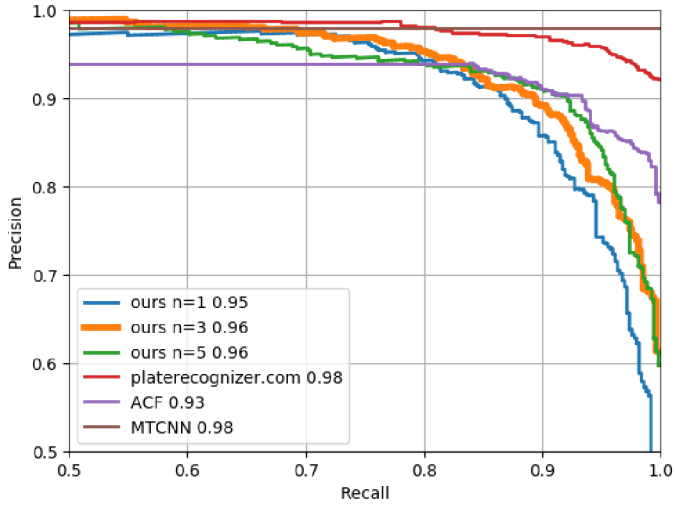

Figure 6.5: Recall-Precision characteristics for the proposed method (for different n) and comparison with other methods.

Table 6.4: Tracking evaluation on testing sequences.

Seq.	#tracked	#missed	#false	Det. rate
1	197	1	59	0.995
2	103	1	43	0.990
3	101	5	15	0.953
4	181	1	26	0.995
5	205	3	23	0.986
6	53	0	2	1.000
Total	**840**	**11**	**168**	**0.987**

with precision around 0.8, and recall 98% with precision 0.7 which is sufficient for real-world applications.

In the *Tracking* mode, we detect and track LPs in video sequences and evaluate the detection rate. Here, we require each license plate to be hit at least once in sequence. From the application point of view, the *Tracking* mode is more important. In the test, we use six sequences, each approximately 10 minutes long taken at ten frames per second. The tracking accuracy is summarized in Table 6.4. The total detection rate 98.7 %. Missed license plates are in most cases, those with extreme deformations or very dirty, false alarms are license plate-like patterns in the image (various signs, etc.), see Figure 6.6 for a few examples.

6.5 Conclusions

We presented a method for reliable real-time detection and localization of license plates observed from a moving vehicle, and its FPGA implementation suitable for integration into a smart camera. Our solution is unique, since it uses machine learning-based detection method, it does not require external memory, and enables for real-time frame rates on low-end FPGA. In this paper, we focused on the detection of license plates. However,

Figure 6.6: **(top)** Examples of detected license plates and detector responses. **(bottom)** Missed license plates (annotations marked by red).

the method is more general and can be adapted for detection of other objects too. The proposed solution is capable of handling complex projections of the license plates, such as deformations caused by perspective projection, scaling, and rotations. The proposed solution uses multiple, in our case three, classifiers to ensure the quality of the results while keeping the performance high. The classifiers are based on simple LBP features which makes it easy to implement them in FPGA. We demonstrated that our hardware solution could process 1280×1024 pixel frames at over 40 FPS while taking only a fraction of resources of low-end FPGA (Zynq Z7020). The accuracy of detection measured on the real-world data is 98 % and 98.7 % when combined with tracking, while producing only a modest amount of false detections.

Future work includes further efficiency improvements, investigation of dependency of quality on freedom of projection of images and sampling of the image frames, and possibly also exploitation of other features and detection mechanisms including hybrid approach with neural networks.

Chapter 7

Applications and Future Work

The results presented in the previous chapters prove the hypothesis that it is possible to create an object detector in hardware that outperforms the state-of-the-art in detection performance, performance/resources ratio and accuracy in selected application tasks. The technological goal of the work has also been fulfilled and the creation of an object detector with reasonable accuracy which can process a FullHD video of more than 15 frames per second and detects small and large objects with limited variability. The advantage is the ability to scale performance and parameters of the detector to the required application and thus reduce the cost of the resulting devices.

The practical application of executed work is relatively broad and can be deployed mostly in various smart cameras. Of the possible applications, such detectors could be applied mainly in transport, industry or security systems, in which the object detection in the image is often required. Examples of practical applications are presented next. The proposed detector managed to improve the system parameters: it decreased cost, power consumption and device size, and increased detection accuracy and performance.

Road and traffic monitoring

Applications of the detector in traffic area are license plates and cars detection, which are fundamental to advanced applications such as counting cars, section speed measurement and searching for stolen vehicles. We have been cooperating for a long time with the manufacturer of such transport systems, the company Camea, which is an industrial partner of the University. The new generation of their traffic cameras are going to use Xilinx Zynq SoC, which allows easy deployment of the proposed detectors. It will lead to a reduced price of devices and less energy consumption. Reducing consumption is essential because these traffic systems are often powered by street lightning and run on battery power during the day. Another advantage is the option of fitting components with an industrial range, which is important in areas with unfavourable climatic conditions, e.g. in Russia.

Another popular field of application is the control of parking in residential areas. The industrial partner of the University, Camea, has recently developed a system for this purpose using cars equipped with cameras. These cars pass through the parking zones and check the individual parking permissions by looking up the license plate registered in the system. The current system has six industrial cameras on the roof of a car to capture all directions. A detection and processing system, built on industrial computers, is located in the trunk of the vehicle. The issue with this solution are huge computational demands for license plate detection, in addition, performed by six cameras. This increases the price of the computer

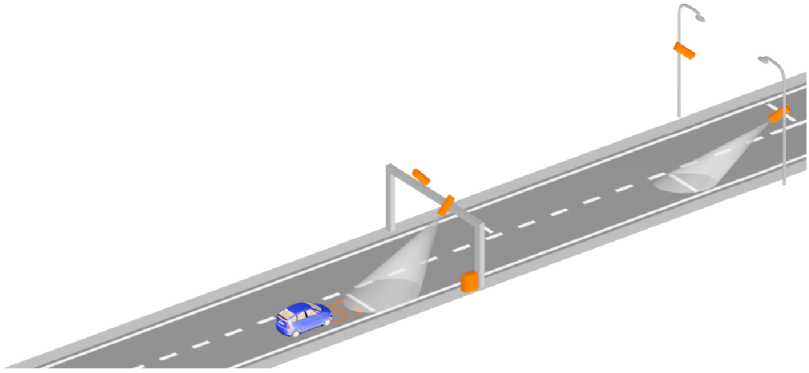

Figure 7.1: System diagram for section speed measurement. Images obtaine from *www.camea.cz*

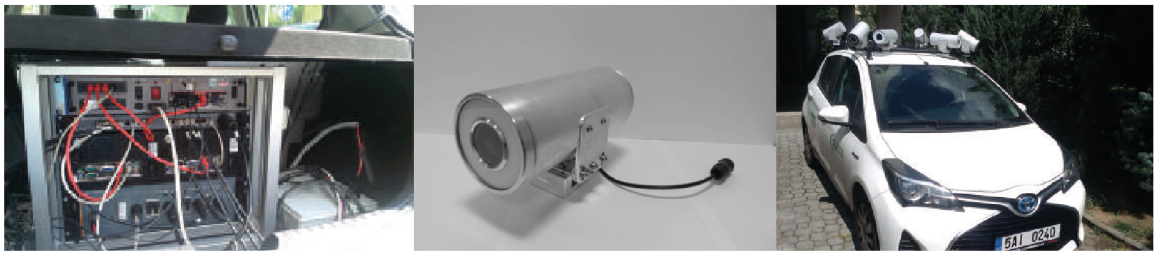

Figure 7.2: Picture on the left shows a powerful computer system for license plate detection and processing installed in the trunk of a car; in the middle, there is a standard industrial camera and on the right a demonstration of the installation of these cameras on a car for parking control in residential areas. Images obtained from *www.camea.cz*

system and causes issues, mainly with cooling and energy consumption. Deployment of the proposed hardware detector directly into the cameras is now being prepared. It will reduce equipment costs, energy consumption and related difficulties with cooling.

Industrial usage

The application of the detector in industry and production is broad. Some examples are the detection of products and components on production line belt, the detection of defects and the detection of markers for the calibration of machinery. The advantages of using detection in hardware in industrial applications are low price and small device size, allowing easy installation to the existing systems.

We tested the detection of defects in quality control of nonwoven production where a high-speed line camera is used. Defects, such as a clogged nozzle or compressed insects, are detected in one resolution on a wide image stripe. The possibility of data storage is limited due to high throughput. The proposed detector connected to the cascade made it possible to process high throughput data and achieved very high accuracy.

Research projects

During my doctoral studies, I participated in several research projects funded by the European Union and the Czech Technological Agency (TAČR). The EU funded projects include CRAFTERS, Almarvi, EMC2 and FitOptiVis. These projects target smart integration

Figure 7.3: The camera prototype using OnSemi CMOS image sensor and Xilinx Zynq SoC.

and optimisation technologies for efficient image and video processing systems. The TAČR funded projects V3C, CEPTIS and AITIV, were focused on embedded computing platforms for optical inspection in the industry and transport applications.

The designed detectors were often used in these research projects to create a prototype device. For example, a prototype smart camera has been created to detect the license plate in low light conditions, using a proposed detector, multi-exposure scanning and HDR image processing. We provided the detector component (IP cores) with source codes to our industrial and academic partners in the EMC2 and FitOptiVis projects to use them in their applications.

Future work

In future work, I would like to follow up on this topic and work on the applicability of the proposed solution in practice. I want to continue working on what has been done in the research projects mentioned in this section. I want to focus on the use of the detector in the security area. I can also see a potential benefit in combining object detection and high dynamic range image processing, for example, when detecting objects in bad light conditions. Přibyl et al. [138] reported that feature point detection on multi-exposure and HDR images achieves better accuracy than using the single-exposure. A similar accuracy improvement can be expected for object detection. Therefore, we collected a data set of multi-exposure images from traffic applications, where complicated light conditions, such as sharp backlight and high contrast, often occur. First experiments confirm an improvement in detection accuracy, which is promising for further research in this area.

Another option that improves the accuracy of detectors in hardware is the use of multi-channel features similar to ACF. Currently, we are working on an ACF based hardware detector. The ACF detector uses the same sequential engine as the detectors presented in this work. The difference is that image features are pre-calculated in memory, compared to the current solution where they are calculated directly from the image when evaluating weak classifiers. Related increased memory requirements are reduced by quantizing feature values to a few bits as suggested by Mitsunari et al. [46] and by aggregating neighbouring values to lower resolutions. Dividing the stored features into multiple memory banks enables to parallelize the calculation of weak classifiers, leading to increased performance. Due to a tree structure of the classifiers, the selection of a memory bank for evaluating one weak classifier is dependent on the previous feature value. This leads to memory access conflicts. Mitsunari

et al. manage these conflicts by elaborating many positions simultaneously and planning the evaluation dynamically to use as many banks as possible simultaneously. Dynamic planning for multiple banks is very complex and demanding for computing resources. We proposed a method of constraining access to individual banks directly during the training process. This allows statical scheduling of a parallel evaluation of weak classifiers resulting in full use of all banks without increasing the resource requirements. Experiments show that the constraints do not reduce the accuracy of the detector. The proposed ACF-based hardware detector outperforms the others in the performance and accuracy of detection while maintaining an excellent power-resource ratio. The results will be published in the near future.

Nowadays, CNN based detection methods achieve the best accuracy. Their massive deployment in FPGAs is only limited by high demands on computing resources. However, technological progress increases the number of transistors on the chip while maintaining the price. It can be assumed that this will allow a much larger deployment of CNN in hardware in the future, to the extent that it could replace most of the other methods.

The combination of boosted classifier based detector and CNN classifier for post-processing detection increases detection accuracy and mainly decreases the amount of false-positive detections. The use of ACF detector to find candidate positions with CNN model to filter out non-object positions is prevalent [137, 139, 140]. For hardware, it is suitable to use SoC, where the boosted detector is evaluated on the FPGA, and the candidate positions are filtered on the CPU or GPU. The number of candidate positions is low, therefore, the performance requirements of the processor or GPU is small. The resulting system would have comparable accuracy to CNN solution, moreover cheaper cost and power consumption. In the combination of boosted detectors and CNN classifiers, I perceive a potential, and I want to investigate it further.

Chapter 8

Conclusion

The main goal of the research conducted in this thesis was to deeply investigate methods for optimising the object detector in images running on FPGA. The newly proposed methods enable creating an object detector in hardware that outperformed state-of-the-art in better detection performance, better performance/resources ratio and better accuracy in selected application tasks. The object detector has been developed using FPGA solely and tested on the face and license plates detection.

The proposed detectors use boosted soft cascades of classifiers with local image feature as weak classifiers. The combination of the unique structure of memory and the local features enabled the effective sequential evaluation of weak classifiers. Parallel processing of multiple independent positions in the image significantly increased detection performance. Cascade connection allows to distribute the calculation among multiple detectors optimally and thus scale the performance and resources consumption to the specific application. The newly designed method enables an efficient multi-scale detection on-the-fly without the use of external memory and large FPGA memory requirements.

The presented scientific contribution aids to create an object-in-image hardware detector usable in practice. A significant benefit is the scalability of performance and resource consumption. It rends possible to develop a detector, which can process either a FullHD video at 60 fps on FPGA with a current price of approximately 100 USD or an HD video at the same speed on a chip with a third of the price. It also allows to process images in high resolution; the proposed hardware detector is the first presented solution for the detection of all-sized objects, even tiny, at 4K resolution. In terms of accuracy, the proposed detector achieves better results than other similar detectors. On the comparative face detection dataset, it achieved 97% accuracy compared to only 91% being the best result until then.

The proposed detectors are expected to be utilised in smart cameras in industrial, transport or security applications, i.e. in tasks such as the detection of faces, pedestrians, products, licence plates. In this work, the practical use is demonstrated on the task of license plate detection for parking control in residential areas. The proposed solution has many advantages over the existing system, such as lower power consumption leading to decrease of heating, as well as lower cost and smaller size of the resulting system.

The use of boosted classifiers for object detection in hardware still remains a reasonable approach. Neural networks are the state-of-the-art in non-hardware object detection; however their deployment on FPGA has enormous resource requirements. In the future, I can see a benefit in combining boosted detectors with CNN to improve the accuracy of detection while maintaining reasonable demands on computing resources and the cost of devices. Another possible way to improve accuracy could be the use of multi-channel features such as ACF.

Bibliography

[1] P. Zemcik, R. Juranek, P. Musil, M. Musil, and M. Hradis, "High performance architecture for object detection in streamed videos," in *Field Programmable Logic and Applications (FPL)*, 2013.

[2] P. Musil, R. Juránek, M. Musil, and P. Zemčík, "Cascaded stripe memory engines for multi-scale object detection in fpga," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 1, pp. 267–280, 2019.

[3] P. Musil, R. Juránek, and P. Zemčík, "Unconstrained license plate detection in hardware," in *International Conference on Vehicle Technology and Intelligent TransportSystems (VEHITS)*, 2021 under review.

[4] D. A. Forsyth and J. Ponce, *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[5] Z. Jeffrey and S. Ramalingam, "High definition licence plate detection algorithm," in *Southeastcon*, 2012.

[6] K. Horak, J. Klecka, and P. Novacek, "License plate detection using point of interest detectors and descriptors," in *Telecommunications and Signal Processing (TSP)*, June 2016, pp. 484–488.

[7] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley Publishing, 2009.

[8] B. Leibe, A. Leonardis, and B. Schiele, "Robust object detection with interleaved categorization and segmentation," *Int. J. Comput. Vision*, vol. 77, no. 1-3, pp. 259–289, 2008.

[9] O. Chum and A. Zisserman, "An exemplar model for learning object classes," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[10] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, 2003, pp. II–II.

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *CVPR*, 2001.

[12] L. Bourdev and J. Brandt, "Robust object detection via soft cascade," in *CVPR*, 2005.

[13] J. Šochman and J. Matas, "Waldboost - learning for time constrained sequential detection," in *CVPR*, 2005.

[14] E. Ohn-Bar and M. M. Trivedi, "To boost or not to boost? on the limits of boosted trees for object detection," *CoRR*, vol. abs/1701.01692, 2017.

[15] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[16] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[17] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018.

[18] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016.

[19] V. N. Vapnik, *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[20] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *Int. J. Comput. Vision*, vol. 38, pp. 15–33, June 2000.

[21] J. Mutch and D. G. Lowe, "Object class recognition and localization using sparse features with limited receptive fields," *International Journal of Computer Vision (IJCV)*, vol. 80, no. 1, pp. 45–57, October 2008.

[22] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR*, 2005.

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

[24] M. Wang and Z. Zhang, "Fpga implementation of hog based multi-scale pedestrian detection," in *2018 IEEE International Conference on Applied System Invention (ICASI)*, 2018, pp. 1099–1102.

[25] J. Dürre, D. Paradzik, and H. Blume, "A hog-based real-time and multi-scale pedestrian detector demonstration system on fpga," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2018)*, 02 2018, pp. 163–172.

[26] X. Du, M. El-Khamy, J. Lee, and L. S. Davis, "Fused DNN: A deep neural network fusion approach to fast and robust pedestrian detection," *CoRR*, vol. abs/1610.03466, 2016.

[27] M. Ilas, "Hog algorithm simplification and its impact on fpga implementation: With applications in car detection," in *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2017, pp. 1–6.

[28] Y. Zhou, Z. Chen, and X. Huang, "A pipeline architecture for traffic sign classification on an fpga," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 950–953.

[29] H.-C. Lai, M. Savvides, and T. Chen, "Proposed fpga hardware architecture for high frame rate face detection using feature cascade classifiers," in *BTAS*, 2007.

[30] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using haar classifiers," in *FPGA*, 2009.

[31] J. Granát, A. Herout, M. Hradiš, and P. Zemčík, "Hardware acceleration of adaboost classifier," in *Workshop on Multimodal Interaction and Related Machine Learning Algorithms (MLMI)*, 2007, pp. 1–12.

[32] M. Hiromoto, H. Sugano, and R. Miyamoto, "Partially parallel architecture for adaboost-based detection with haar-like features," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, 2008.

[33] C. Kyrkou and T. Theocharides, "A flexible parallel hardware architecture for adaboost-based real-time object detection," in *VLSI Systems*, 2011.

[34] C. Huang and F. Vahid, "Scalable object detection accelerators on fpgas using custom design space exploration," *SASP*, 2011.

[35] B. Brousseau and J. Rose, "An energy-efficient, fast fpga hardware architecture for opencv-compatible object detection," in *Field-Programmable Technology (FPT)*, 2012.

[36] J. Sochman and J. Matas, "Adaboost with totally corrective updates for fast face detection," in *FGR*, 2004, pp. 445–450.

[37] X. Baro, S. Escalera, J. Vitria, O. Pujol, and P. Radeva, "Traffic sign recognition using evolutionary adaboost detection and forest-ecoc classification," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 1, pp. 113–126, 2009.

[38] E. Yousefi, A. H. Nazem Deligani, J. Jafari Amirbandi, and M. Karimzadeh Kiskani, "Real-time scale-invariant license plate detection using cascade classifiers," in *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, 2019, pp. 399–402.

[39] M. S. Juwad, Y. Li, and S. Abdulla, "Ensemble of adaboost cascades of 3l-lbps classifiers for license plates detection with low quality images," *Expert Systems with Applications*, vol. 92, 09 2017.

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

[41] H. Jiang and E. G. Learned-Miller, "Face detection with the faster R-CNN," *CoRR*, vol. abs/1606.03473, 2016.

[42] S. Jin, D. Kim, T. T. Nguyen, D. Kim, M. Kim, and J. W. Jeon, "Design and implementation of a pipelined datapath for high-speed face detection using fpga," *IEEE Transactions on Industrial Informatics*, vol. 8, pp. 158 – 167, 2012.

[43] F. Kadlček and O. Fučík, "Automatic synthesis of small adaboost classifier in fpga," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2013.

[44] P. Zemčík and M. Žádník, "Adaboost engine," in *FPL*, 2007.

[45] K. Mitsunari, J. Yu, and M. Hashimoto, "Hardware architecture for fast general object detection using aggregated channel features," in *2018 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2018, pp. 55–58.

[46] K. MITSUNARI, J. Yu, T. Onoye, and M. Hashimoto, "Hardware architecture for high-speed object detection using decision tree ensemble," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E101.A, pp. 1298–1307, 09 2018.

[47] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight yolov2: A binarized cnn with a parallel support vector regression for an fpga," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 31–40.

[48] Y. Ma, T. Zheng, Y. Cao, S. Vrudhula, and J.-s. Seo, "Algorithm-hardware co-design of single shot detector for fast object detection on fpgas," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, 2018.

[49] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. Lee, "A high-throughput and power-efficient fpga implementation of yolo cnn for object detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861–1873, 2019.

[50] H. Kang, "Real-time object detection on 640x480 image with vgg16+ssd," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 419–422.

[51] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance cnn processor based on fpga for mobilenets," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 136–143.

[52] C. Yeh, C. Lin, K. Muchtar, H. Lai, and M. Sun, "Three-pronged compensation and hysteresis thresholding for moving object detection in real-time video surveillance," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 6, pp. 4945–4955, 2017.

[53] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and efficient fpga-based feature detection employing the surf algorithm," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 3–10.

[54] Y. Sato and Y. Kuriya, "Multi-scale elastic graph matching for face detection," *Journal on Advances in Signal Processing*, vol. 2013, p. 175, 11 2013.

[55] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[56] P. Dollar, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," in *Proceedings of the British Machine Vision Conference*. BMVA Press, 2009, pp. 91.1–91.11, doi:10.5244/C.23.91.

[57] P. Dollar, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 8, p. 1532–1545, Aug. 2014.

[58] A. Haar, "Zur theorie der orthogonalen funktionensysteme," *Mathematische Annalen*, vol. 69, pp. 331–371, September 1910.

[59] A. Herout, R. Jošth, P. Zemčík, and M. Hradiš, "Gp-gpu implementation of the „local rank differences" image feature," in *Proceedings of International Conference on Computer Vision and Graphics 2008*, ser. Lecture Notes in Computer Science. Springer Verlag, 2008, pp. 1–11.

[60] T. Maenpaa and M. Pietikainen, "Multi-scale binary patterns for texture analysis," in *SCIA03*, 2003, pp. 885–892.

[61] P. Zemcik, M. Hradis, and A. Herout, "Local rank differences - novel features for image," in *Proceedings of SCCG 2007*, 2007, pp. 1–12.

[62] M. Hradiš, A. Herout, and P. Zemčík, "Local rank patterns - novel features for rapid object detection," in *Proceedings of International Conference on Computer Vision and Graphics 2008*, ser. Lecture Notes in Computer Science, 2008, pp. 1–2.

[63] L. Polok, A. Herout, P. Zemčík, M. Hradiš, R. Juránek, and R. Jošth, "„local rank differences" image feature implemented on gpu," in *Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, ser. Lecture Notes In Computer Science; Vol. 5259. Springer Verlag, 2008, pp. 170–181.

[64] R. Juránek, M. Hradiš, and P. Zemčík, *Real-Time Systems*. InTech Education and Publishing, 2012, ch. Real-Time Object Detection with Classifiers, p. 21.

[65] C. Kyrkou, C.-S. Bouganis, T. Theocharides, and M. M. Polycarpou, "Embedded hardware-efficient real-time classification with cascade support vector machines," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, pp. 99–112, 2016.

[66] P. Dollár, "Piotr's Computer Vision Matlab Toolbox (PMT)," https://github.com/pdollar/toolbox.

[67] H. Song, B. Jeong, H. Choi, T. Cho, and H. Chung, "Hardware implementation of aggregated channel features for adas," in *2016 International SoC Design Conference (ISOCC)*, 2016, pp. 167–168.

[68] S. Martelli, D. Tosato, M. Cristani, and V. Murino, "Fast fpga-based architecture for pedestrian detection based on covariance matrices," in *Image Processing (ICIP)*, 2011.

[69] Y. Yazywa, T. Yashimi, T. Tsuzuki, T. Dohy, Y. Yamauchi, T. Yamashita, and H. Fujiyoshi, "Fpga hardware with target-reconfigurable object detector," *IEEE Transactions on Information and Systems(IEICE)*, vol. 98, no. 9, pp. 1637–1645, 2015.

[70] Y. Said and M. Atri, "Efficient and high-performance pedestrian detector implementation for intelligent vehicles," *Intelligent Transport Systems(IET)*, vol. 10, pp. 438–444, 2016.

[71] M. Bilal, A. Khan, M. U. Karim Khan, and C. Kyung, "A low-complexity pedestrian detection framework for smart video surveillance systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 10, pp. 2260–2273, 2017.

[72] F. An, P. Xu, Z. Xiao, and C. Wang, "Fpga-based object detection processor with hog feature and svm classifier," in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*, 2019, pp. 187–190.

[73] J. Li, X. Liu, F. Liu, D. Xu, Q. Gu, and I. Ishii, "A hardware-oriented algorithm for ultra-high-speed object detection," *IEEE Sensors Journal*, vol. 19, no. 10, pp. 3818–3831, 2019.

[74] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.

[75] S. Ghaffari, P. Soleimani, K. F. Li, and D. W. Capson, "Analysis and comparison of fpga-based histogram of oriented gradients implementations," *IEEE Access*, vol. 8, pp. 79 920–79 934, 2020.

[76] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin, and W. Wolf, "Embedded hardware face detection," in *17th International Conference on VLSI Design. Proceedings.*, 2004, pp. 133–138.

[77] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. 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*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161–170.

[78] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[79] V. Jain and E. Learned-Miller, "Fddb: A benchmark for face detection in unconstrained settings," University of Massachusetts, Amherst, Tech. Rep. UM-CS-2010-009, 2010.

[80] B. Zhang, J. Li, Y. Wang, Y. Tai, C. Wang, J. Li, F. Huang, Y. Xia, W. Pei, and R. Ji, "Asfd: Automatic and scalable face detector," 2020.

[81] W. Kienzle, G. Bakir, M. Franz, B. Schölkopf, and W. Y, "Face detection - efficient and rank deficient," *Advances in Neural Information Processing Systems, 673-680 (2005)*, vol. 17, 01 2005.

[82] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 304–311.

[83] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, 2016.

[84] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *CoRR*, vol. abs/1702.03044, 2017.

[85] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[86] D. Barina, P. Musil, M. Musil, and P. Zemcik, "Single-loop approach to 2-d wavelet lifting with jpeg 2000 compatibility," in *2015 International Symposium on Computer Architecture and High Performance Computing Workshop (SBAC-PADW)*, 2015, pp. 31–36.

[87] P. Zemčík, P. Musil, and M. Musil, *High Dynamic Range Video*. Elsevier Science, 2016, pp. 145–154.

[88] S. Nosko, M. Musil, P. Musil, and P. Zemčík, "True hdr camera with bilateral filter based tone mapping," in *SCCG '17: Spring Conference on Computer Graphics 2017*. Association for Computing Machinery, 2017, pp. 1–9.

[89] P. Zemčík, R. Juránek, P. Musil, M. Musil, and M. Hradiš, "High performance fpga object detector: Hardware prototype," in *2013 23rd International Conference on Field programmable Logic and Applications*, 2013, pp. 1–1.

[90] D.-K. Kim, J.-H. Jung, T. T. Nguyen, D.-J. Kim, M.-S. Kim, K.-H. Kwon, and J.-W. Jeon, "An fpga-based parallel hardware architecture for real-time eye detection," *JSTS*, 2012.

[91] K. Curran, X. Li, and N. M. Caughley, "The use of neural networks in real-time face detection," *Journal of Computer Science*, 2005.

[92] L. Zhang, R. Chu, S. Xiang, S. Liao, and S. Z. Li, "Face detection based on multi-block lbp representation," in *ICB*, 2007.

[93] A. Herout, P. Zemčík, M. Hradiš, R. Juránek, J. Havel, R. Jošth, and M. Žádník, *Pattern Recognition Recent Advances*. IN-TECH, 2010, ch. Low-Level Image Features for Real-Time Object Detection.

[94] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*. Washington, DC, USA: IEEE Computer Society, 1998, p. 555.

[95] R. Juránek, A. Herout, and P. Zemčík, "Impelementing local binary patterns with simd instructions of cpu," in *Proceedings of Winter Seminar on Computer Graphics*. West Bohemian University, 2010, p. 5.

[96] X. Cui, Y. Liu, S. Shan, X. Chen, and W. Gao, "3d haar-like features for pedestrian detection," in *ICMCS*, 2007.

[97] D. Goshorn, J. Cho, R. Kastner, and S. Mirzaei, "Field programmable gate array implementation of parts-based object detection for real time video applications," *International Conference on Field Programmable Logic and Applications (FPL 2010)*, pp. 582 – 587, August 2010.

[98] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[99] A. Herout, R. Jošth, R. Juránek, J. Havel, M. Hradiš, and P. Zemčík, "Real-time object detection on cuda," *Journal of Real-Time Image Processing*, vol. 2010, no. 1, pp. 1–12, 2010.

[100] C. Kyrkou, C. Ttofis, and T. Theocharides, "Fpga-accelerated object detection using edge information," *International Conference on Field Programmable Logic and Applications (FPL 2011)*, pp. 167 – 170, September 2011.

[101] T. Kryjak, M. Komorkiewicz, and M. Gorgon, "Fpga implementation of real-time head-shoulder detection using local binary patterns, svm and foreground object detection," in *Design and Architectures for Signal and Image Processing (DASIP)*, 2012.

[102] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *ISCAS*, 2010.

[103] Z. Xu, R. Shi, Z. Sun, Y. Li, Y. Zhao, and C. Wu, "A heterogeneous system for real-time detection with adaboost," in *High Performance Computing and Communications*, 2016.

[104] J. Yang, Y. Yang, Z. Chen, L. Liu, J. Liu, and N. Wu, "A heterogeneous parallel processor for high-speed vision chip," *IEEE TSCVT*, 2017.

[105] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[106] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[107] S. Bhattarai, A. Madanayake, R. J. Cintra, S. Duffner, and C. Garcia, "Digital architecture for real-time cnn-based face detection for video processing," in *CCAA*, June 2017, pp. 1–6.

[108] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*, ser. FPGA '16. New York, NY, USA: ACM, 2016.

[109] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in *ISLPED*. New York, NY, USA: ACM, 2016, pp. 326–331.

[110] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *Int. J. Comput. Vision*, vol. 73, no. 2, pp. 213–238, 2007.

[111] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Gray scale and rotation invariant texture classification with local binary patterns," in *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part I.* London, UK: Springer-Verlag, 2000, pp. 404–420.

[112] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Mach. Learn.*, vol. 37, no. 3, pp. 297–336, 1999.

[113] K. Turkowski, "Properties od surface-normal transformations," in *Graphics Gems*, 1990.

[114] J. Sochman and J. Matas, "Learning fast emulators of binary decision processes," *International Journal of Computer Vision*, vol. 83, no. 2, pp. 149–163, June 2009.

[115] C. Caraffi, T. Vojir, J. Trefny, J. Sochman, and J. Matas, "A System for Real-time Detection and Tracking of Vehicles from a Single Car-mounted Camera," in *ITS Conference*, Sep. 2012, pp. 975–982.

[116] A. Broggi, E. Cardarelli, S. Cattani, P. Medici, and M. Sabbatelli, "Vehicle detection for autonomous parking using a soft-cascade adaboost classifier," in *2014 IEEE IVSP*, June 2014.

[117] A. A. Biyabani, S. A. Al-Salman, and K. S. Alkhalaf, "Embedded real-time bilingual alpr," in *Communications, Signal Processing, and their Applications (ICCSPA)*, 2015.

[118] P. S. Ha and M. Shakeri, "License plate automatic recognition based on edge detection," in *2016 Artificial Intelligence and Robotics (IRANOPEN)*, April 2016, pp. 170–174.

[119] X. Zhai, F. Bensaali, and S. Ramalingam, "Real-time license plate localisation on fpga," in *CVPR 2011 WORKSHOPS*, June 2011, pp. 14–19.

[120] S. Chhabra, H. Jain, and S. Saini, "Fpga based hardware implementation of automatic vehicle license plate detection system," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sept 2016, pp. 1181–1187.

[121] M. S. Khalil and F. Kurniawan, "License plate detection method for real-time video of low-cost webcam based on hybrid svm-heuristic approach," in *Information Technology: New Generations (ITNG)*, 2014.

[122] S. Y. Yang, Y. C. Lu, L. Y. Chen, and D. C. Cherng, "Hardware-accelerated vehicle license plate detection at high-definition image," in *2011 First International Conference on Robot, Vision and Signal Processing*, Nov 2011, pp. 106–109.

[123] Y. Yuan, W. Zou, Y. Zhao, X. Wang, X. Hu, and N. Komodakis, "A robust and efficient approach to license plate detection," *IEEE Transactions on Image Processing*, vol. 26, pp. 1102 – 1114, 2017.

[124] A. Elbamby, E. E. Hemayed, D. Helal, and M. Rehan, "Real-time automatic multi-style license plate detection in videos," in *Computer Engineering Conference (ICENCO)*, 2016.

[125] G. Hsu, J. Chen, and Y. Chung, "Application-oriented license plate recognition," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 2, pp. 552–561, 2013.

[126] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, V. Loumos, and E. Kayafas, "A license plate-recognition algorithm for intelligent transportation system applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 3, pp. 377–392, Sept 2006.

[127] V. Mai, D. Miao, R. Wang, and H. Zhang, "An improved method for vietnam license plate location," in *2011 International Conference on Multimedia Technology*, July 2011, pp. 2942–2946.

[128] S.-Z. Wang and H.-J. Lee, "Detection and recognition of license plate characters with different appearances," in *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, vol. 2, Oct 2003, pp. 979–984 vol.2.

[129] G. A. M. Sborz, G. A. Pohl, F. Viel, and C. A. Zeferino, "A custom processor for an fpga-based platform for automatic license plate recognition," in *2019 32nd Symposium on Integrated Circuits and Systems Design (SBCCI)*, 2019, pp. 1–6.

[130] C. Arth, H. Bischof, and C. Leistner, "TRICam – an embedded platform for remote traffic surveillance," in *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*, June 2006, pp. 125–125.

[131] L. Xie, T. Ahmad, L. Jin, Y. Liu, and S. Zhang, "A new cnn-based method for multi-directional car license plate detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 507–517, 2018.

[132] C. L. P. Chen and B. Wang, "Random-positioned license plate recognition using hybrid broad learning system and convolutional networks," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2020.

[133] F. Gao, Y. Cai, Y. Ge, and S. Lu, "Edf-lpr: a new encoder decoder framework for license plate recognition," *IET Intelligent Transport Systems*, vol. 14, no. 8, pp. 959–969, 2020.

[134] X. WU, J. QIU, and A. QIU, "An efficient license plate location algorithm based on deep leaming," in *2020 International Conference on Computer Engineering and Application (ICCEA)*, 2020, pp. 543–546.

[135] S. M. Silva and C. R. Jung, "License plate detection and recognition in unconstrained scenarios," in *2018 European Conference on Computer Vision (ECCV)*, Sep 2018, pp. 580–596.

[136] E. Yousefi, A. H. Nazem Deligani, J. Jafari Amirbandi, and M. Karimzadeh Kiskani, "Real-time scale-invariant license plate detection using cascade classifiers," in *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, 2019, pp. 399–402.

[137] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, p. 1499–1503, Oct 2016.

[138] B. Přibyl, A. Chalmers, P. Zemčík, L. Hooberman, and M. Čadík, "Evaluation of feature point detection in high dynamic range imagery," *Journal of Visual Communication and Image Representation*, vol. 38, pp. 141 – 160, 2016.

[139] A. Verma, R. Hebbalaguppe, L. Vig, S. Kumar, and E. Hassan, "Pedestrian detection via mixture of cnn experts and thresholded aggregated channel features," in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, 2015, pp. 555–563.

[140] Z. Ma and P. Gao, "Research on the cascade pedestrian detection model based on ldcf and cnn," in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2018, pp. 314–320.