FPGA-ACCELERATED SLIDING WINDOW CLASSIFIER WITH STRUCTURED FEATURES

Ondřej Sychrovský, Martin Matoušek, Radim Šára

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague
166 27, Prague 6, Technická 2, Czech Republic
{sychro1,xmatousm,sara}@cmp.felk.cvut.cz

ABSTRACT

Certain classification tasks in computer vision require the classifier response to be computed in every pixel of an image. When combined with large, complex features, it becomes challenging to build such a classifier on a standard PC architecture and achieve real-time performance.

We present an FPGA implementation of a car wheel classifier response computation, built as an instantiation of a generic classification system. An interesting optimization problem concerning performance and speed is addressed. Our implementation is running in real-time as a part of a more complex collision mitigation system based on car detection in video data.

1. INTRODUCTION

A typical visual object detection task requires "computing something everywhere in the image". One of the simplest forms uses a set of classifiers, one per primitive object, to compute response maps in all image pixels. This works as an operator transforming the image to another image, more suitable for building an object detector.

We want to detect passenger cars in videos of crossing traffic based on a structured model that includes the wheels, the wheelbase, the pillars etc. The wheel classifier and detector, Fig. 1, is therefore the part which has to be run everywhere in the image. When evaluating the detection likelihood ratio for the rest of the model, the raw input image is needed but it is not necessary to access all its pixels. We concentrate in this paper on this approach – to pre-compute wheel likelihood maps for use in a car detector.

We present an FPGA implementation of an object classifier. Our solution is generic and can be used in most localized object detection tasks. In addition, to boost performance under diverse illuminations, our procedure includes learnable adaptive image re-quantization based on local image contrast. We refer the reader to [1] for a more complete picture of the whole car detection subsystem and to [2] for an extended version of this paper.

In recent years, the need for real-time applications of object detection has risen, and many hardware systems have been proposed for accelerating the object classification and detection. Most common is the AdaBoost-based classifier adapted from the classification scheme by Viola and Jones for face detection [3]. Majority of the papers use Haar-like features and focus on FPGA implementation of the integral image feature extraction and cascaded classification. Gao et al. [4] have implemented a 40-stage cascade on an FPGA, processing 16 features per stage simultaneously. Hiromoto et al. [5] compute features in the first few stages in parallel in order to speed up the initial decision process.

He et al. [6] also use Haar features, but they build a cascade of Artificial Neural Network classifiers. Support Vector Machines are also suitable for FPGA acceleration, as in [7], where again a cascading approach is used. Huge amount of data necessary to process in traditional sliding window approaches is reduced in [8], where a hardware edge detector has been added to steer the focus of the classification cascade.

We utilize the AdaBoost-based framework for detection. This learning scheme has an advantage that a criterion for selecting efficient features can be simply employed.

We place emphasis on using more structured features than simple Haar-like ones; general kernels that require full convolution to be computed. The properties of the bank of kernels are given by an application, and can represent some knowledge about the detected objects that can be hard to learn automatically. In our application, we use rotationally invariant templates for wheel detection.

We do not employ a classification cascade because we use fewer (but more complicated) features. The second rea-

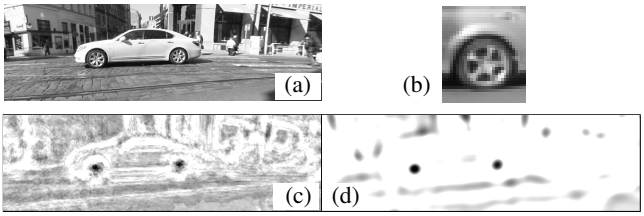


Fig. 1: Motivation: the wheel detector. (a) The input image. (b) An example of a positive sample. (c) The FPGA output—the classifier response map. (d) The map post-processed by spatial aggregation.

son is that we need the classifier response map (not the decision) for the entire image to make post-processing (Fig. 1).

2. CLASSIFIER WITH STRUCTURED FEATURES

A linear classifier classifies fixed-size image patches $\mathbf{S}_{x,y}$ around a pixel (x, y). The classifier learned by AdaBoost is composed of a set of *weak classifiers*, each one is defined as a triplet $(\mathbf{M}_i, t_i, g_i)$, where $\mathbf{M}_i$ is its associated kernel (either real or complex matrix), t_i is a threshold and $g_i = \pm 1$ is a sign. A kernel has the same size as an image patch.

For a classifier of car wheels, rotational invariance is a natural requirement to avoid the need to learn all possible wheel rotations and risk overfitting. We construct some of the kernels as a complex sinusoid with unit L_2 norm. The modulus of the dot product with such a kernel is then a rotationally invariant feature.

Specifically, for an image patch $\mathbf{S}_{x,y}$ the dot product with the kernel $d_i = \mathrm{vec}(\mathbf{S}_{x,y})^\top \mathrm{vec}(\mathbf{M}_i)$ is compared with the threshold to obtain the weak classifier's decision y_i,

$$v_i = \begin{cases} d_i & \text{if } \mathbf{M}_i \in \mathbb{R}^{w\times h}, \\ |d_i| & \text{if } \mathbf{M}_i \in \mathbb{C}^{w\times h}, \end{cases} \quad y_i = \begin{cases} +1 & \text{if } g_i v_i > t_i, \\ -1 & \text{if } g_i v_i \le t_i. \end{cases} \quad (1)$$

These decisions are aggregated using learned weighting coefficients α_i to form the overall classifier response

$$r(\mathbf{S}_{x,y}) = \sum_{i=1}^{n} \alpha_i y_i. \quad (2)$$

To simplify the FPGA implementation, the kernel values are quantized to $\{-1, 0, +1\}$, allowing multiplications in the dot product to be replaced by simpler logical operations, and the L_2 norm in (1) is replaced by L_1 norm,

$$v_i = |\Re(d_i)| + |\Im(d_i)| \quad \text{if } \mathbf{M}_i \in \mathbb{C}^{w\times h}. \quad (3)$$

Using a reference PC implementation, we compared a classifier with quantized kernels and L_1 norm to a classifier with kernels quantized to 8 bits and L_2 norm, and proved that this change has very little effect on the classification error.

For the image patch size 25×29 pixels, we have learned a classifier of wheels consisting of 55 unique kernels, Fig. 2 (the complex ones have two parts, so there are 77 parts in total) and 150 weak classifiers (some of them share the same kernel). Both limits were fixed for the learning phase.

3. FPGA IMPLEMENTATION

We have implemented the classifier on a Xilinx ML605 evaluation board with a PCIe DMA data transfer wrapper.

The processing pipeline implementing general functionality of a classifier is summarized in Fig. 3. The classifier is specified by supplying numeric constants. We automatically generated the appropriate parts of VHDL source.

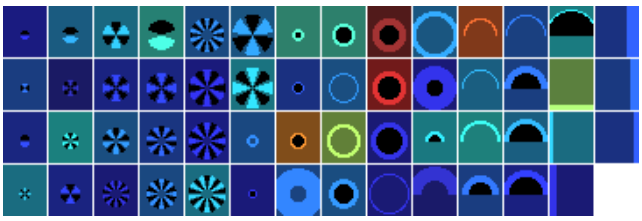
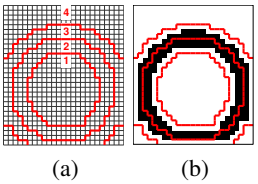


Fig. 2: The selected kernels. All fan-like kernels are complex, their real parts are shown. The kernel values are encoded as -1 = black, 0 = darker, $+1$ = brighter.



(a) (b)

	slice number			
	1	2	3	4
krnl (b)		•	•	

(c)

Fig. 4: Slices. (a) Both the 25×29 image patch and all kernels have been divided into four slices (with borders shown as thick red lines). (b) An example kernel (black pixels mean nonzero values) occupying slices 2 and 3. (c) Kernel-Slice Incidence Table. A single row in the table represents a single kernel. The '•' marks slices (columns) with non-zero values. The kernel (b) occupies slices 2 and 3.

3.1. Linecache

The sliding window buffer outputs the whole image patch of dimensions 25×29. This block is implemented as a shift register, consisting of both general flip-flops and BRAMs.

3.2. Slice Selector and Batches

This section describes generic algorithmic optimizations that are needed for real-time performance. This, together with Sec. 4, is the main contribution of this paper.

The number of dot products (77) is larger than the number of dot product-computing blocks (DPCBs, Sec. 3.4) we are able to place to the FPGA (30). Also, processing 25×29 image patch at once is hard to implement efficiently because

- too many parallel routes (25×29×n bits) would prove achieving the timing closure very difficult,
- there are not enough HW resources, e.g. BRAMs that are used to store the kernels,
- some kernels cover only a sub-region of the image patch; processing speed can benefit from some way of discarding the pixels not covered by a kernel.

We have divided the image patch area into four slices, 182 pixels each. To exploit circularity of most kernels, we have set the slices to be circular as well, Fig. 4. Since the whole image patch has already been cached, the slice selector can select pixels completely arbitrarily.

The number of dot products is addressed by computing them in batches evaluated in a sequence. Each batch computes several dot products in parallel. The distribution of kernels into batches is discussed later in Sec. 4.

The slice selector block outputs one of the four slices at each clock cycle in the predefined order that is specific for

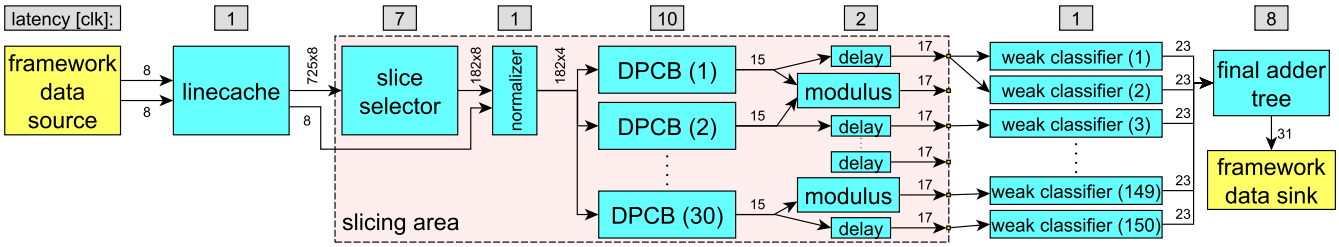


Fig. 3: The processing pipeline. Along with the data wires, there are also *valid* signals that are used to mark valid data between blocks. The values in gray rectangles show the latency in clock cycles. For the used clock frequency of 125 MHz, the total latency is 240 ns.

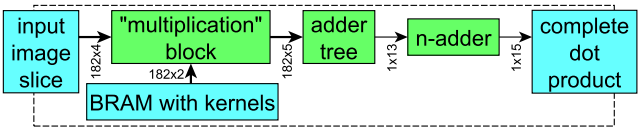


Fig. 5: The dot product-computing block (DPCB) overview.

the selected set of kernels and slice shapes.

3.3. Normalizer

An adaptive image patch normalization reduces the sensitivity of the classifier to local scene illumination. Each pixel in the image slice is multiplied by a coefficient and the result is re-quantized to 4 bits. These 182 multiplications are done in dedicated DSP blocks. The coefficients for each image patch are computed in advance on the host PC based on the mean intensity of the patch using real-time integral image technique.

3.4. Dot Product-Computing Block (DPCB)

The block, Fig. 5, takes slices of the image patch, one at a time, and produces a dot product of the entire image patch with a single kernel. Because the kernel values are limited to $\{-1, 0, +1\}$, a case selection is performed instead of multiplication. A slice sub-product is computed in adder tree and sequentially summed in n-adder to get the result.

There are 30 DPCBs in the FPGA fabric. They all have the same input but each computes a different dot product.

3.5. Modulus Block

This block computes the L_1 norm (3) of complex kernels, for real-valued kernels the data are simply passed through.

3.6. Weak Classifier and Adder Tree

The weak classifier outputs the $\alpha_i y_i$ term in (2). Several weak classifiers share the same kernel dot product. We have put all the 150 weak classifier blocks in the FPGA fabric. Weak classifiers' parameters α_i, t_i, g_i, are hardwired during synthesis. The final adder tree sums up all the $\alpha_i y_i$ terms.

4. KERNEL SLICING SCHEME

The Kernel-Slice Incidence Table (Fig. 4c) defines slices used in computation for each kernel. A particular slice containing all zero values is discarded from computation.

krnl	slice number 1	2	3	4
1	•			
	•			
2	•			
	•			
3	•			
	•			
4	•			
	•			
5	•			
	•			
6	•			
	•			
7	•			
	•			
8	•			
	•			
23	•			
24	•			
25	•			
26	•			
27	•			
29	•			
39	•			
	-			
	-			
	-			
	-			
	-			
	-			
	-			
req. FPs total: 23				

krnl	slice number 1	2	3	4
9	•	•		
	•	•		
10	•	•		
	•	•		
11	•	•		
	•	•		
12	•	•		
	•	•		
13	•	•		
	•	•		
14	•	•		
	•	•		
15	•	•		
	•	•		
16	•	•		
	•	•		
17	•	•		
	•	•		
30	•	•		
31	•	•		
32	•	•		
33	•	•		
34	•	•		
41	-	•		
	-	-		
	-	-		
	-	-		
	-	-		
	-	-		
req. FPs total: 24				

krnl	slice number 1	2	3	4
18	•	•	•	-
	•	•	•	-
19	•	•	•	-
	•	•	•	-
20	•	•	•	-
	•	•	•	-
21	•	•	•	-
	•	•	•	-
22	•	•	•	-
	•	•	•	-
28	•	•	•	•
35	•	•	•	-
36	-	•	•	-
37	-	•	•	•
38	•	•	•	-
40	-	•	•	•
42	-	•	•	-
43	-	•	•	-
44	•	•	•	-
45	-	-	•	-
46	•	•	•	-
47	•	•	•	-
48	•	•	•	•
49	•	•	•	•
50	-	•	•	•
51	-	-	•	•
52	-	•	•	•
53	•	•	•	•
54	-	•	•	•
55	-	•	•	•
req. FPs total: 30				

Table 1: Possible batch assignment constructed from the Incidence Table for the set of filters in Fig. 2 and 30 DPCBs. There are slices containing nonzero entires ('•'), slices with all zeros that are either processed and discarded ('-') or not processed at all (' '). The slice processing sequence is $(1 \mid 1, 2 \mid 1, 2, 3, 4)$. In total, the 30 DPCBs evaluate $7 \cdot 30 = 210$ slices, while only 158 of them (the number of '•' marks) contain nonzero data, i.e. the utilization is 75%. Note that both the real and the imaginary parts of a complex-valued kernel must be processed at the same time—the doubled rows must not be separated.

As described in Sec. 3.2, the kernels are grouped into batches that are run in a sequence. The set of N_i^B slices processed in a particular batch i is a union of slices required by all kernels in that batch (that have to use the same slices). The total number of slices $N = \sum_i N_i^B$ processed in the sequence then determines the overall processing time. Grouping the kernels that require similar set of slices helps minimizing the N. See Tab. 1 for example of grouping.

To summarize, the Kernel Slicing Scheme is defined by

1. the *number and shape of slices,*
2. the *Kernel-Slice Incidence Table,* and
3. the *distribution of kernels to batches.*

num. blocks	clk cycles	LUTs	BRAMs	DSPs
30	7	73202	371	364
20	10	53445	261	364
13	15	39577	184	364
10	19	32456	151	364

Table 2: Resource consumption for different number of DPCBs for the image patch of size 25×29. Each block adds 11 BRAMs and about 2000 LUTs.

	overall			per DPCB	
patch size	LUTs	BRAMs	DSPs	LUTs	BRAMs
9×15	16803	67	68	369	1
15×19	26506	111	144	731	2
19×25	38506	177	238	1297	7
25×29	53445	261	364	1952	11
29×35	76541	347	508	2720	15

Table 3: Resource consumption for different image patch sizes with fixed number of 20 DPCBs.

Currently, we have learned the classifier independently, manually created the slice shapes and used semi-manual exhaustive search for distributing the kernels into batches. In Tab. 1 it is clearly visible, that the results are not optimal. One can possibly replace the kernels to make the overall code length N shorter while keeping the classifier performance or more kernels can be added to use the free DPCBs, thus keeping the processing time but improving the classifier performance. This can be easily incorporated into a weak feature selection method during the AdaBoost learning.

All parts of the Kernel Slicing Scheme, as well as the classifier itself, should be optimized with respect to the overall detection rate and processing time, given the kernel bank, the training data for our classifier and the hardware constraints. This is a challenging yet unsolved problem.

5. PERFORMANCE, SCALING AND LATENCY

The latency for a single pixel is 240 ns as has been shown in Fig. 3. The processing time depends on the size of the input image. Since we spend $N = 7$ clock cycles processing a single pixel, for a 951×400 pixel image we currently have cycle time of $951 \cdot 400 \cdot 7/125\,\mathrm{MHz} = 21.3\,\mathrm{ms}$ per image which determines the effective latency.

Our proposed architecture is scalable, allowing to adjust the number of DPCBs to suit the size of a given FPGA, Table 2. Changing the size of the image patch also greatly influences the design, Table 3. On the other hand, the size of the input image has almost no influence on the chip layout.

It proved useful to use two slice selectors instead of one, each with different set of slices, and to distribute the DPCBs between them. This brings yet another level of complexity to the optimization problem from Sec. 4. We have tested this design, using the slice sequence codes $(1 \mid 1 \mid 1, 2, 3, 4)$ and $(1 \mid 1, 2 \mid 1, 2, 3)$, i.e., reducing the length from 7 to 6 slices.

6. CONCLUSIONS

We have proposed an approach for implementing a dense linear classifier on an FPGA. The top-level scheme is chosen as quite generic. The specific instantiation is then tuned to a particular application. An interesting optimization problem has been discussed, that would allow to efficiently choose a trade-off between processing speed and classification performance.

The proposed architecture is scalable, it is up to the designer how many blocks are needed to place in the FPGA.

The performance evaluation proved that the design is able to run fast, making it ideal for use in real-time applications. The proposed wheel classifier response map computation is used in a car detector running in an intelligent vehicle as a part of a more complicated collision mitigation system [1], that requires processing cycle of 20–30 fps and a maximum latency of 200 ms.

ACKNOWLEDGEMENT

This work was supported by the European Commission under interactIVe, a large scale integrated project, part of the FP7-ICT-246587 for Safety and Energy Efficiency in Mobility. The authors would like to thank all partners within interactIVe for their support. Special thanks go to Stefan Wonneberger at Volkswagen Group Research.

REFERENCES

[1] P. Heck, J. Bellin, M. Matoušek, S. Wonneberger, O. Sychrovský, R. Šára, and M. Maurer, "Collision mitigation for crossing traffic in urban scenarios," in *Proc IEEE Intelligent Vehicles Symposium*, 2013.

[2] O. Sychrovský, M. Matoušek, and R. Šára, "FPGA-accelerated sliding window classifier with structured features," Center for Machine Perception, Czech Technical University, Prague, Research Report CTU–CMP–2013–16, June 2013.

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

[4] C. Gao and S.-L. Lu, "Novel FPGA based Haar classifier face detection algorithm acceleration," in *Proc Int Conf on Field-Programmable Logic and Applications*, 2008, pp. 373 –378.

[5] 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, no. 1, pp. 41–52, 2009.

[6] C. He, A. Papakonstantinou, and D. Chen, "A novel SoC architecture on FPGA for ultra fast face detection," in *IEEE International Conference on Computer Design*, 2009, pp. 412–418.

[7] M. Papadonikolakis and C. Bouganis, "Novel cascade FPGA accelerator for support vector machines classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 7, pp. 1040 –1052, July 2012.

[8] C. Kyrkou, C. Ttofis, and T. Theocharides, "FPGA-accelerated object detection using edge information," in *Proc Int Conf on Field Programmable Logic and Apps*, 2011, pp. 167 –170.