Michał FULARZ, Marek KRAFT
POZNAN UNIVERSITY OF TECHNOLOGY, INSTUTUTE OF CONTROL AND INFORMATION ENGINEERING
3a Piotrowo St., 60-965 Poznań

Hardware implementation of a decision tree classifier for object recognition applications

Abstract

Hardware implementation of a widely used decision tree classifier is presented in this paper. The classifier task is to perform image-based object classification. The performance evaluation of the implemented architecture in terms of resource utilization and processing speed are reported. The presented architecture is compact, flexible and highly scalable and compares favorably to software-only solutions in terms of processing speed and power consumption.

Keywords: decision tree, hardware implementation, FPGA, object recognition.

1. Introduction

Decision trees are among the most popular approaches used for classification. This conceptually simple method, based on successive partitioning of the instance space, has so far been successfully applied to a wide range of tasks [1][2]. Moreover, classification performance of decision trees can be further improved by using multiple trees in conjunction, forming an ensemble classifier [3].

In this paper, hardware implementation of a decision tree is described. While the presented application of the classifier is image-based object recognition based on uniform local binary pattern features, the architecture is general and can be used in a broad range of other classification tasks. The training of the classifier is performed on a PC. Upon the completion of the training, the structure of the tree is extracted and translated into a ready to implement description in VHDL. The resulting solution is a compact, low power, relatively good performance classifier, which can be used as a standalone solution or as a part of a bigger system.

2. The implemented algorithms

The input data to the decision tree-based classifier are concatenated histograms of uniform local binary patterns (ULBPs). Thus, the computation of the feature vector begins with the computation of the basic LBPs for each pixel in the image. The LBP is formed based on gray level value comparison operations in 8-neighborhood as shown in Fig. 1.

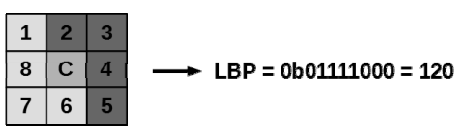

Fig. 1. Illustration of the basic LBP computation process

If the intensity value of a pixel in the 8-neighborhood is greater than that of the central pixel C, the corresponding bit in a 8-bit vector is set to '0', otherwise it is set to '1'. The resulting 8-bit binary value is then converted to a decimal number, giving rise to 256 different LBP variants. Reduction of the number of possible LBP variants is made possible by using their uniform variants (uniform LBPs – ULBPs), that is the LBPs are ones with less than three '0'-'1' or '1'-'0' transitions in the binary vector. As the ULBPs carry most of the information necessary for successful classification [4], the number of unique descriptors can be reduced from 256 to 59 with no significant impact on distinguishability.

However, individual ULBP values are not enough to perform successful detection and/or classification. The most common approach for the construction of a descriptor of an image region is to divide the region into rectangular cells and compute feature occurrence histograms in these cells. The histograms from the cells are then concatenated and form the complete descriptor. To detect/classify an object in the image, a window of the size of the descriptor region slides over the whole image. The division of the window into individual cells allows capturing the spatial arrangement of feature histograms that is characteristic for a given class of objects. The illustration of the complete descriptor formation process is shown in Fig. 2.

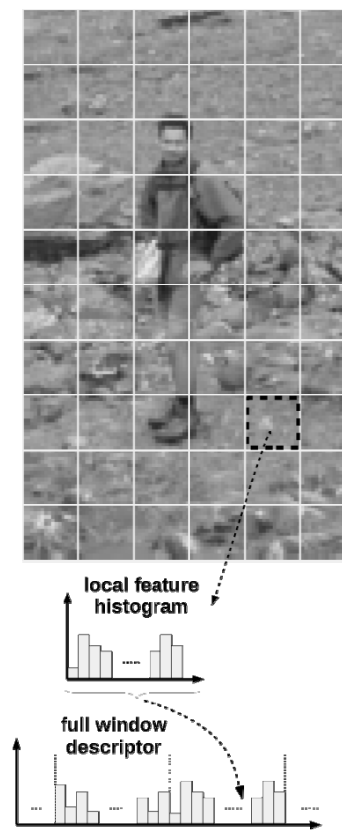

Fig. 2. Illustration of the descriptor formation procedure

Complete descriptors computed from image regions are passed to the decision tree classifier.

Decision trees can be used for both classification and regression and can be constructed as a result of support supervised, non-parametric learning [1, 2]. The goal of learning is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The constructed decision trees are easy to be understood and visualized. The input data can come in both numerical and categorical form and does not require any special treatment or preparation.

The decision tree training was performed using a scikit-learn machine learning library for the Python programming language [5]. The goal of the training was the recognition of human silhouette. The training data was taken from the dataset published in the paper by Dalal et al [6]. Images of humans and non-humans

with the resolution of 96 × 160 pixels (an example is shown in Fig. 2) were used. Each detection window was divided into a 6 × 10 cell grid, with each cell having the size of 16 × 16 pixels.

3. Description of the implemented architecture

The implemented decision tree module is designed to work with an LBP-based image region descriptor processor [7]. The block diagram of both cores implemented in a Xilinx Zynq device is shown in Fig. 3. The processor subsystem (two ARM Cortex A9 processors) is responsible for controlling the dataflow, visualization and communication. The processing pipeline uses the AXI4-Stream infrastructure and can accept the input data (stream of images) from an image sensor or from an external DDR memory through DMA (only the first option is shown in Fig. 3.). The LBP module calculates the ULBPs for each pixel, computes the local histograms for each cell and then aggregates the information into a complete descriptor by concatenating the individual histograms. The image region descriptor is used by the decision tree module as an input vector. The module outputs the information if the currently analyzed region contains a human silhouette. All the operations are fully pipelined and done in one clock cycle. The results are stored in the memory for higher level processing (e.g. tracking of detected objects).

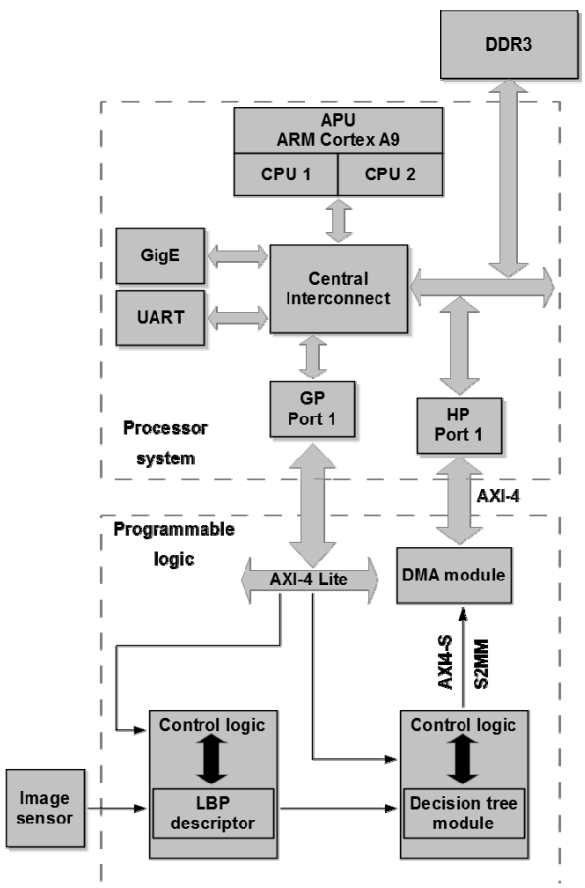

Fig. 3. Block diagram of the whole system

The block diagram of the decision tree module is shown in Fig. 4. The module is created based on the trained classifier – all the splits (comparisons), leaves and paths (list of splits) from root to each leaf are extracted. In contrast to the software implementation, all the comparison operations are performed, but they are executed in parallel in one clock cycle. This can be seen in the figure as the first stage of operations. The second stage uses the output from the previous step to establish the object class (human silhouette or not). This is done using the extracted information about the leaves by analyzing all the comparison results proceeding the analyzed leaf. With all the necessary data from the preceding stages ready, this stage also takes one clock cycle. As all the paths to the leaves are exclusive, only one will be true and it will be used to determine the object class.

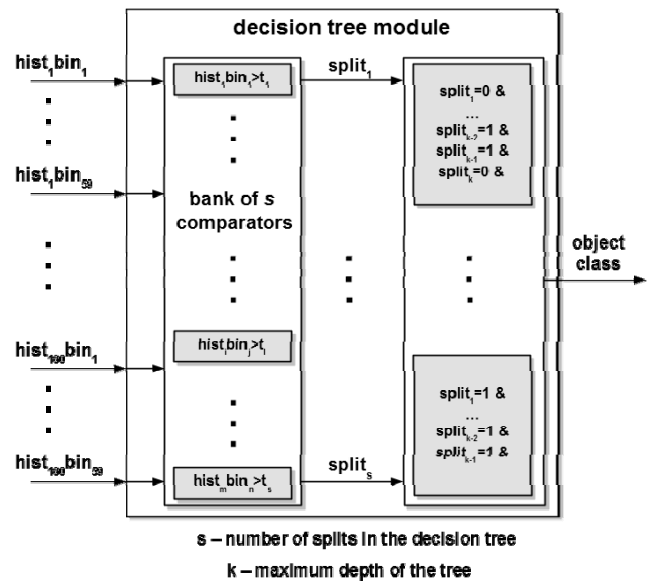

Fig. 4. Block diagram of the decision tree module

Since hand-coding the implemented decision tree module in a hardware description language would be cumbersome, a Python script, which automates this task was developed. The script performs an analysis of the decision tree object created by the classifier learning in scikit-learn and extracts all the leaves alongside all the splits. The extracted information is used to generate two VHDL processes – one for each stage of the hardware module shown in Fig. 4. The tests performed on dataset data confirm the correctness of the classifier operation generated using the script.

All the aforementioned tools and the procedures for preparing the dataset, training and testing the decision tree are freely available at Github repository [9].

4. Results and discussion

The utilization of programmable logic resources necessary to implement a number of variants of the decision tree classifier is summarized in Table 1.

Tab. 1. FPGA resource utilization and the number of leaves and splits based on the tree depth (designations: FFs - flip-flops, LUTs - lookup tables)

Tree depth	Number of splits	Number of leaves	FFs	LUTs
5	30	31	31	64
7	54	55	55	119
10	65	66	66	133
15	77	78	78	161
20	83	84	84	179

The implementation remains compact, even if the size of the tree is relatively large – it is below 0,5% of a mid-range XC7Z020 Zynq device for the biggest, 20-level deep tree. The features used for making the decision in each split are 8-bit numerical values (histogram bins). The implementation of the tree classifier consists therefore of a set of relatively simple comparators and additional logic for testing the root to leaf paths in the tree based on conjunction of comparison results. It is worth noting that even though the presented architecture is very simple, the classification

accuracy is well over 80% for a rather difficult problem. The described solution can be used as a part of a larger system or a standalone coprocessor. Its simplicity allows the implementation of the classifier in resource-constrained applications. Moreover, multiple simple tree-based classifiers can be combined into a more complex, ensemble classifier like random forests.

Thanks to the use of fast DMA-based communication interfaces, the classifier in conjunction with the coprocessor for LBP descriptor computation [7] is capable of operation on a continuous stream of input image data. The clock frequency of the combined circuit is 75 MHz, enabling classification of 75 million samples per second using the sliding window approach. To compare, a PC with Intel Core i5-4570 CPU clocked at 3.2 GHz requires 13.32 us per single classification alone (not including the image processing involved), being an order of magnitude slower (approx. 75 thousands samples per second).

5. Conclusions and future work

The paper presents hardware implementation of a decision tree classifier in a programmable logic device. The resulting solution is scalable, easily configurable and capable of very fast operation, while consuming little power. Moreover, the solution is easy to use, as the code for the hardware implementation is automatically generated based on the results from a popular software library used for scientific computations. The source code for the software is available for download. The solution is planned to be used as one of the image processing modules of an embedded smart camera [8].

Future work will be focused on a more detailed analysis of the performance of the presented architecture on a range of data from multiple application fields, e.g. signal processing, image processing, communication networks etc. A study on further extension of the presented solution into an ensemble of classifiers is also planned.

This research was financed by the Polish National Science Centre grant funded according to the decision DEC-2011/03/N/ST6/03022, which is gratefully acknowledged.

6. References

[1] Rokach L., Maimon O.: Top-down induction of decision trees classifiers - a survey. IEEE Transaction on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 35, pp 476-487, 2005.

[2] Quinlan J.R.: Decision trees and decision-making. IEEE transactions on Systems. Man and Cybernetics, vol. 20, pp. 339-346, 1990.

[3] Breiman L.: Random forests. Machine learning, vol. 45, pp. 5-32, 2001.

[4] Ojala T., Pietikäinen M. and Mäenpää T.: Multiresolution Gray-scale and Rotation Invariant Texture Classification with Local Binary Patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24(7), pp. 971-987, 2002.

[5] Pedregosa F.: Scikit-learn: Machine learning in Python. The Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[6] Dalal N., Triggs B.: Histograms of oriented gradients for human detection. In Proc. Of Int. Conf. on Computer Vision and Pattern Recognition CVPR 2005, vol. 1, pp. 886-893, 2005.

[7] Kraft M., Fularz M.: A Hardware Architecture for Calculating LBP-Based Image Region Descriptors, to appear in Proc. of the 9th Int. Conf. on Computer Recognition Systems (accepted for publication).

[8] Fularz M., Kraft M., Schmidt A., Kasiński A.: The Architecture of an Embedded Smart Camera for Intelligent Inspection and Surveillance. Advances in Intelligent Systems and Computing, vol. 350, pp. 43-52, 2015.

[9] https://github.com/Michal-Fularz/decision_tree

Received: 07.04.2015 Paper reviewed Accepted: 02.06.2015

Michał FULARZ, MSc, eng.

Graduated from the Poznan University of Technology in 2010 specializing in Robotics. Since then he has been employed at the Institute of Control and Information Engineering of the Poznan University of Technology. His research focuses on accelerating compute-intensive algorithms using FPGAs, image processing, embedded systems and robotics.

e-mail: michal.fularz@put.poznan.pl

Marek KRAFT, PhD, eng.

Graduated from Poznan University of Technology in 2005. He received the PhD degree from the same University in 2013. In the same year he has been appointed as an assistant professor at the university's Institute of Control and Information Engineering. His research interests are computer vision, embedded systems, robotics, parallel processing and high performance computing.

e-mail: marek.kraft@put.poznan.pl