Embedded Face Detection Application based on Local Binary Patterns

Laurentiu Acasandrei
Instituto de Microelectrónica de Sevilla
IMSE-CNM-CSIC
Sevilla, Spain
laurentiu@imse-cnm.csic.es

Angel Barriga
Instituto de Microelectrónica de Sevilla/Univ Sevilla
IMSE-CNM-CSIC/Univ. Sevilla
Sevilla, Spain
barriga@imse-cnm.csic.es

Abstract—**In computer vision during the recent years a new paradigm for object detection has stimulated researchers and designers interest. The foundation of this new paradigm is the Local Binary Pattern (LBP) which is a nonparametric operator that efficiently extracts the features of local structures in images. This communication describes a software embedded implementation of LBP based algorithm for object detection, in particular targeting frontal face detection.**

Keywords-Face detection; LBP; embedded software

I. Introduction

Face detection is an important aspect for biometrics, video surveillance and human computer interaction. Detection systems require huge computational and memory resources due to the complexity of detection algorithms. An interesting operator for face detection is the Local Binary Patterns (LBP) [1]. As a non parametric operator, LBP efficiently extracts the features of local structures in images by comparing each pixel with its neighboring pixels. Initially LBP was proposed for texture analysis but due to it two most important properties (tolerance regarding monotonic illumination changes and computational efficiency) it evolved to an adequate mechanism for feature extraction and classification. Nowadays LBP is employed into a multitudes of fields like: face detection, face recognition, facial expression analysis, detection and recognition of lesions in different human organs, etc.

Typically the LBP based object detectors required a large amount of computational resources and it needs huge memory bandwidth. Other object detectors algorithms based on Support Vector Machines, Neural Networks or Viola-Jones are more computationally expensive. For some applications, e.g. real-time video face detection or vehicle plate identification, software-only embedded systems fail to reach the expected performances. To overcome this limitations an embedded flexible solution have been explored.

The structure of the paper is the following. The next section describes briefly the face detection architecture based on LBP. Section III shows the embedded implementation of the face detection system. Then the algorithm optimization is described. Finally, in section V will be discussed the time performance of the proposed system.

II. Face Detection: LBP

The LBP operator proposed in [1] is a gray-scale and rotation invariant texture operator based on local binary patterns. By comparing the gray scale value of pixel with the gray scale values of the local neighbors pixels situated in a circular symmetric proximity, an LBP operator can be invariant to the gray scale variations. Rotation invariance is achieved by determining the fixed set of rotation invariant patterns that the LBP operator incorporates. An example that shows how LBP code is generated is illustrated in Fig. 1. For a 3x3 image pixel, the central pixel is compared with neighboring pixels. If the value of a neighbor pixel is greater than the central pixel it is encoded with 1, otherwise with 0. For each pixel, the LBP binary code is obtained by concatenating all these binary values in a clockwise direction, which starts from its top-left neighbor. If the objects are larger than 3x3 pixels, the LBP operator cannot capture the dominant feature of the objects properly. To overcome this limitation the LBP operator was later generalized to use neighborhoods of different sizes (see Fig. 2).

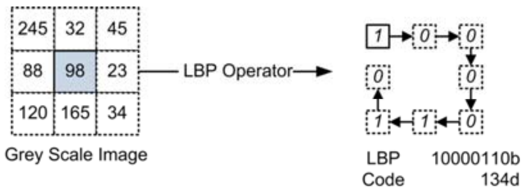


Figure 1. Example of basic LBP operator

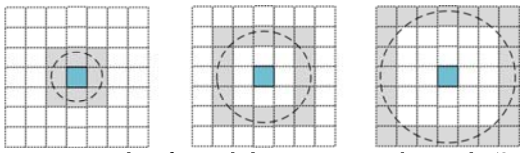
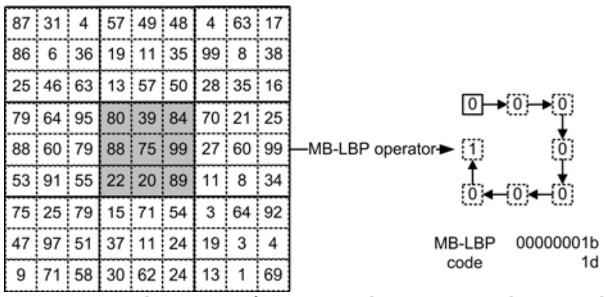


Figure 2. Examples of extended LBP operator. The circular(8,1), (16,2) and (24,3) neighborhoods

Reference [2] proposes a new representation for the non parametric operator: Multi-scale Block LBP (MB-LPB). Compared to the basic LPB, in MB-LBP, the computation is done based on average values of block subregions, instead of individual pixels (Fig. 3). It results that the MB-LBP has several advantages: a) is more robust than LBP, b) it provides a more complete image representation than the basic LBP, by capturing the micro and macro information from a subregion, and c) it can be computed very efficiently using integral images.



87	31	4	57	49	48	4	63	17
86	6	36	19	11	35	99	8	38
25	46	63	13	57	50	28	35	16
79	64	95	80	39	84	70	21	25
88	60	79	88	75	99	27	60	99
53	91	55	22	20	89	11	8	34
75	25	79	15	71	54	3	64	92
47	97	51	37	11	24	19	3	4
9	71	58	30	62	24	13	1	69

Figure 3. Example MB-LBP for a 9x9 pixel region. For each 3x3 pixel sub-region, or block, the average image intensity is computed. The neighboring block sums are compared with the sum of central sum thus obtaining the MB-LBP code

The grey scale intensity of each block, or sub-region, can be efficiently calculated using integral image. The integral image consists of accumulating for every pixel the value of the previous pixels:

$$I_{P(x,y)} = \sum_{i'\leq i,\, j'\leq j} p(i',j') \qquad (1)$$

In basic LBP [1], a uniform pattern subset of LBP code is used to improve the performance. An LBP patterns is called uniform if it contains at most two bitwise transitions from 0 to 1 or vice versa when the corresponding bit string is considered circular. However, in [2] the original definition of uniform LBP patterns, based on the transition of pixel values, cannot be used for MB-LBP. This is due to the fact that same properties of circular continuities in 3x3 patterns are not valid when the size of LBP block becomes larger. In order to build efficient classifiers the redundant information must be removed. For it a simple and efficient classifier is built by means of AdaBoost learning algorithm [3]. This allows the selection of a small number of visual characteristics from a very large set of potential characteristics.

The architecture of the classifier is made by combining simple classifiers in cascade (Fig. 4). The detection technique is based on exploring the image by means of a window searching for features. The search window is scaled in order to locate faces of different sizes. First stages of the detector cascade consist of simple, very fast and low cost detectors. As a consequence those regions that do not contain faces are rapidly rejected and only region with stronger face candidates are accepted. The following detectors increase in complexity in order to perform a more detailed analysis on a smaller set of areas of interest. Faces are detected at the end of the cascade.

This work was supported in part by Spanish Ministerio de Ciencia y Tecnología under the Project TEC2011-24319, by Spanish Ministerio de Economía y Competitividad under the Project IPT-2012-0695-390000, and by Consejo Superior de Investigaciones Científicas under the Project 201450E034. Co-financed by FEDER.

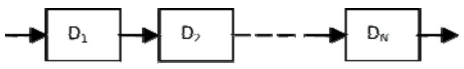


Figure 4. Detectors cascade architecture.

III. Implementation of Face Detection on Embedded System

The MB-LBP object detection algorithm was implemented in OpenCV [4] as an object detection application for human faces. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing.

For the MB-LBP algorithm the OpenCV library contains a set of pre-trained classifiers. Even more, the OpenCV provides a training application that permits the creation and training of classifiers for new objects. The robustness and the accuracy of the classifier depend on the training dataset and training parameters. For face detection the classifier is composed of 139 MB-LBP operators organized in 20 cascaded stages. The MB-LBP operators, used for features extraction, are trained to be applied for squared image regions of 24x24 pixels. This means, that initially, the LBP cascade is trained to detect faces from 24x24 pixel regions. Unfortunately, in practice the faces have arbitrary sizes in an image. Detecting objects, i.e. faces, having arbitrary sizes can be done in two modes. In the first mode the image is step-by-step scaled down, using for example linear interpolation, until it reaches a predefined minimal dimension. For each scaled version of the image an exhaustive search for faces is performed on consecutive regions. In the second mode the image has fixed dimension and the classifier is step-by-step scaled. To be more precise the size of the MB-LBP operators, i.e. feature extractors, are scaled progressively until they reach a predefined maximum size (typically close to the size of the image). For each scaled version of the classifier and exhaustive search for faces is performed on consecutive regions.

Unfortunately all the functionality of MB-LBP face detection is contained by member functions belonging to different OpenCV classes. If one needs to run the MB-LBP baseline face detection it needs to install the entire OpenCV library on operating system. To use the MB-LBP face detection in an embedded system it has been considered that the majority of embedded environments are capable of running C/C++ applications with or without OS support. This means that the resulting application code has to be compatible for both C, C++ compilers and in the same time to be platform independent. Also most of the Systems on Chip (SoC's) have no floating point support. Thus it has been decided to use integer operations instead of floating point operations in order to preserve the generality of the application for the embedded system world. The steps underwent in porting the application are: 1) Making the application stand-alone and removing the OpenCV library dependencies; 2) Replacing classes with structures; 3) Building a C++ application that reads the MB-LBP classifier cascaded from an XML file and saves them in a simpler format in C header file; 4) Using software performance analysis tools (Valgrind [5]) to identify and examine the bottlenecks; 5) Replacing all the floating point operations with integer operations and floating point data with integer data (it was found that for scaling with 20 bits (precision of 20 bits for the floating point decimals) the integer and floating point applications obtain identical results); 6) The resulted application that uses only integer operations was compiled for LEON3 processor where it was successfully verified; 7) Building a GUI software tool that runs on a PC and is responsible of sending configuration data, command, and raw images to the LEON3 SoC in order to measure the performance of the MB-LBP embedded application for different configurations.

At end it was obtained: a) A standalone MB-LBP face detection application; b) A ported standalone MB-LBP face detection application targeting LEON3 SoC;

IV. Algorithm Optimization

The standalone MB-LBP face detection application was analyze using the Gprof, Gcov, Valgrind tools. The measurements of software application, running on PC with Ubuntu Linux, reveal some interesting facts. The measurements were done on VGA size (640x480 pixels) images. It was noticed that more than 96% of the execution time was consumed by applying the classifier cascade on all image regions and for all scaled images. Only 3.11% of the execution is spent in scaling the image to different dimensions and only 0.5% is used to calculate the integral representation for the scaled images.

On the other hand, the MB-LBP face detection algorithm works on a large amount of image data and so the following pertinent question is born: how much of the application time is spent by the processor/hardware in executing instructions (arithmetical, logical or jump instructions) compared to the data transfers? Only 22.42% of the total time was spent in instruction execution. The majority of the time, i.e. 77.57% is spent in data transfers. Almost 50% of the total time is spent in data read operations compared to the 27.92% used for data write operations. Taking into consideration that MB-LBP feature extraction is applied to data read from system memory, it comes as no surprise that read operations are almost twice more frequent than write operations.

In order to obtain a fast MB-LBP face detection application it was decided to optimize the MB-LBP evaluation mechanism with an emphasis on reducing data bandwidth and accelerating data transfers.

The MB-LBP detection algorithm evaluates a large number of regions in order to find faces. For each region the MB-LBP feature extraction is performed until a conclusion (i.e. face or not a face) is reached. Each feature extraction is accompanied by comparing the intensities of neighboring sub-regions and constructing the LBP code. To efficiently calculate the intensities of all the nine sub-regions (one center sub-region and eight neighbors) the integral image representation is used. We denote with integral pixel $I_{P(x,y)}$ the data contained by the integral image representation at row *x* and column *y*. The $I_{P(x,y)}$ represents the sum of all image pixels enclosed by area having the coordinates $\{0, 0\}$, $\{0, y\}$, $\{x, 0\}$ and $\{x, y\}$. The value of every $I_{P(x,y)}$ is stored on 32 bits and the maximum number of pixels inside a region for which the intensity can be correctly calculated is 2^{24}.

The classifier provided by OpenCV is trained to be applied for 24x24 pixels regions. It results that a sub-region can have a maximum size of 8x8 pixels. Typically the MB-LBP mechanism extracts features from neighboring sub-regions having sizes between 2x2 pixels to 8x8 pixels.

Instead of using the standard integral representation we proposed a confined integral representation. The confined integral pixel $CI^N_{P(x,y)}$ is defined as follows:

$$CI^N_{P(x,y)} = \left\lfloor \sum_{i=0}^{x} \sum_{j=0}^{y} p_{(i,j)} \right\rfloor_N \qquad N > 8 \tag{2}$$

where the pair (x,y) is the coordinate of pixel *P*, *N* is the width in bits of confined integral pixel element and the operator $\lfloor \ \rfloor_N$ truncates the double sum to the *N* least significant bits. The $CI^N_{P(x,y)}$ is similar to the $I_{P(x,y)}$ with the exception that $CI^N_{P(x,y)}$ cannot be greater than 2^N-1. When $CI^N_{P(x,y)}$ overpasses the 2^N-1 it will produce an overflow and only the *N* least significant bits will be taken into consideration. The maximum number of pixels inside a region for which the confined intensity can be correctly calculated is 2^N-8. For a confined integral pixel stored on 64 bits the maximum allowed number of pixels inside a region is $N=2^{56}$, for a confined integral pixel stored on $N=32$ bits the maximum number of allowed pixels inside a region is 2^{24} and for confined integral pixel stored on $N=16$ bits the maximum number of allowed pixels inside a region is 2^8. It results that the standard integral is a particular case of confined Integral where $N=32$.

Taking into consideration that MB-LBP mechanism extracts features from neighboring sub-regions with a maximum size of 8x8 pixel we can use $CI^N_{P(x,y)}$ with $N=16$ to store each confined Integral element on 16 bits. The Confined Integral with $N=16$ obtains the same results as the standard integral for regions having less than 2^8 pixels. Therefore the standard Integral can be safely replaced with Confined Integral$^{N=16}$. A direct consequence is that the memory space needed to store an entire confined Integral is reduced by half, compared to the standard Integral. Unfortunately, for typical 32 bit processors, accessing individual byte or half-word, from the system memory, requires the same amount of time as accessing word data. From a software point of view, running entirely on a 32 bit embedded processor, this will not speed up detection but it will reduce program memory footprint. However, for 16 bit or 8 bit processor architectures, the software using Confined Integral$^{N=16}$ will run faster compared to the software using standard Integral (it needs at least two clock cycles to access a word value).

V. TIME PERFORMANCES OF EMBEDDED MB-LBP

To measure the time performances of the embedded MB-LBP software it was used an embedded system built around LEON3 Sparc V8 processor. The main clock frequency of the on-chip bus (AMBA) and LEON3 processor is 80 Mhz. The processor data cache is composed of 4 sets of 4KB and instruction cache is composed of 2 sets of 8 KB each. The software is loaded and executed on external 256 MB DDR2 memory. For performance comparison three distinct version of the embedded MB-LBP have been measured:

- Ver1 is the ported OpenCV MB-LBP face detection software application to the embedded system
- Ver2 is software accelerated implementation that uses integer numbers.
- Ver3 is the algorithmic accelerated implementation that uses he ported software that uses Confined Integral$^{N=16}$ and only integer numbers.

For a given image, all three embedded software versions evaluate the same regions and give the same detection results. The time measurements and software component profiling was done on a set of nine images representing different information domains: a black and white image, three distinct patterns, a natural landscape image an three distinct images containing one, three and respectively four faces. Each image from the set had three sizes: 240x160, 320x240 and 640x480 pixels.

The measurements were done with the help of the GUI tool created by authors to connect via serial interface with LEON3 SoC. The embedded MB-LBP face detection application running on LEON3 SoC also contains the software driver for the serial interface and data transfer protocol.

The timing results for the Ver1, Ver2 and Ver3 software implementation for different image sizes is presented in Fig. 5. We notice that Ver3 and Ver2 software versions are much faster with ~30% than the OpenCV ported version (Ver1). Unfortunately the algorithm optimized version (Ver3) is not faster than software optimized version (Ver2) because any access (byte, half-word or word) on the AMBA bus is treated as word transfers.

Even if the same number of regions is evaluated for every image, there is an important time difference given by how many stages the MB-LBP classifier needs, for a region, to reach a conclusion (face or no face). Note that because the white and black images have homogenous regions the MB-LBP classifier reaches a faster conclusion. For strongly textured and landscapes images, advanced stages inside the MB-LBP classifier are needed to reach a conclusion, thus resulting in a slower speed. For images containing faces, the speed of the detection is influenced by the number and size of faces found in image. E.g. for the image containing three faces all faces are detected, but for the image containing four faces only three faces are detected because one of the face has profile orientation and the MB-LBP classifier is trained to detected only frontal faces. The LBP evaluation timing usage varies between [58%, 77%] for Ver1, [47%, 69%] for Ver2 and Ver3.

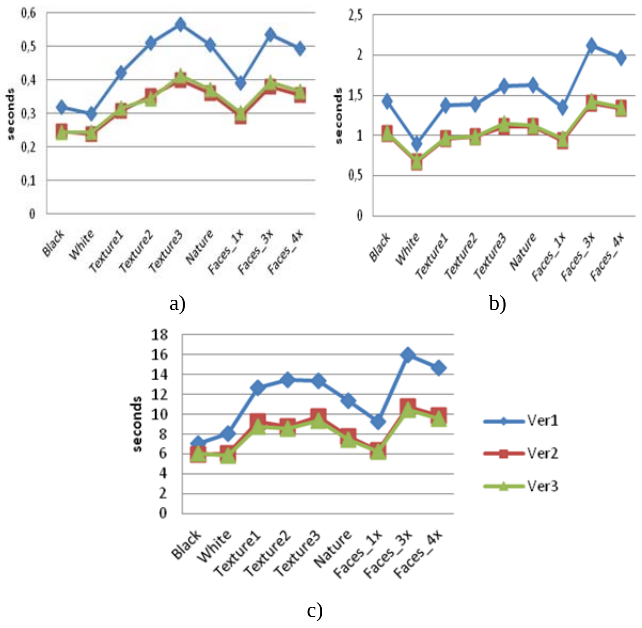


Figure. 5. Embedded MB-LBP software timings. Images of size a) 240x160 pixels, b) 320x240 pixels, c) 640x480 pixels.

In this communication was presented an optimized software embedded implementation of LBP based algorithm for face detection. To reduce the memory footprint and accelerate the detection the algorithm is optimized by using Confined Integral$^{N=16}$ and only integer operations. Analyzing the timing performance of a MB-LBP face application is cumbersome task because is directly influenced by internal structure of the classifier and it dynamic response to heterogonous images. Usually the best case timing performance can be determined with simple homogenous images (black, white or grey). However from a performance point of view, the designer is interested in the maximum detection time. Unfortunately even with a large dataset of random images it unlikely to precisely measure the worst case timing performance of a specific classifier. The authors have observed in practice, for MB-LBP classifier provided by OpenCV, the worst case timing is between 2 to 3 times slower than best case timing. This can be used as a rule of thumb in the initial planning of time budget for a face detection system.

REFERENCES

[1] T. Ojala, M. Pietikainen, and T. Maenpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns”, IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 7, pp. 971–987, Jul. 2002.

[2] S. Liao, X. Zhu, Z. Lei, L. Zhang and S. Z. Li, “Learning Multi-scale Block Local Binary Patterns for Face Recognition”, Int. Conf. on Biometrics, pp. 828-837, 2007.

[3] R.E. Schapire, Y. Freund, P. Bartlett, W.S. Lee, “Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods”, The Annals of Statistics, pp. 1651-1686, 1998.

[4] Open Source Computer Vision (OpenCV): http://opencv.org/

[5] Valgrind dynamic analysis tools : http://valgrind.org