Evolutionary Approximation of Gradient Orientation Module in HOG-based Human Detection System

Michal Wiglasz and Lukas Sekanina
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno, Czech Republic
Email: {iwiglasz, sekanina}@fit.vutbr.cz

***Abstract*—The histogram of oriented gradients (HOG) feature extraction is a computer vision method widely used in embedded systems for detection of objects such as pedestrians. We used Cartesian genetic programming (CGP) to exploit the error resilience in the HOG algorithm. We evolved new approximate implementations of the arctan function, which is typically employed to compute the gradient orientations. When the best evolved approximations are integrated into the SW implementation of the HOG algorithm, not only the execution time, but also the classification accuracy was improved in comparison with the accurate implementation and the state-of-the art approximate implementations.**

***Index Terms*—Functional approximation, Cartesian genetic programming, Histogram of oriented gradients**

I. Introduction

Providing high-quality outputs, for example, in terms of image classification accuracy, is usually computationally expensive and energy demanding. In this paper, we focus on one particular application – human detection in images – implemented on low power devices. Various implementation approaches have been proposed in this area. One of their key distinguishers is a method used to define the so-called features, i.e. the vectors of parameters that represent particular input images in the image classification pipeline. While *hand-crafted features* (such as image gradients) are created by experts and used in application-specific algorithms (such as histogram of oriented gradients, Harr wavelets and shape contexts), *learned features* are derived from data in the process of learning of deep convolutional neural networks. Although learned features achieve better classification accuracy, their usage leads to the 2–4 orders of magnitude overhead in energy consumption as measured on integrated circuits recently developed for embedded vision [1], [2]. If our objective is to minimize the computation cost and energy, we should consider an algorithm based on hand-crafted features, assumed that the quality of output is sufficient for a particular application.

This paper deals with improving of the *histogram of oriented gradients* (HOG) feature extraction method in terms of execution time (and power consumption). The approach is proposed and evaluated in the context of low power pedestrian detection used in driver assistant systems. First, we perform an error resilience analysis of HOG in order to identify such components whose potential inexact implementation will have a minor impact on the quality of result, but will significantly reduce the execution time and power consumption. Then, we design approximate versions of selected component(s) and analyze if our objectives are met. This strategy falls under the umbrella of approximate computing which exploits error resilience of certain applications to optimize their power consumption, execution time and other parameters [3].

In particular, we focus on the gradient orientation module which is typically based on a relatively expensive iterative algorithm computing the arctan function. We employ Cartesian genetic programming (CGP) to automatically evolve a linear code (no loops, no branches) to approximate the gradient orientation module. As the evolved code can immediately be implemented as a combinational circuit, we, in fact, provide a suitable solution not only for software-based implementations, but also for hardware accelerators of HOG. The evolved solution is compared with the original implementation and state-of-the art solutions from the literature, in terms of the execution time and classification accuracy on MIT and INRIA pedestrian datasets.

II. Approximate Computing

Approximate computing was established with the goal of providing more energy efficient, faster, and less complex computer-based systems by allowing some errors in computations [3]. One of the approximation techniques is *functional approximation* whose principle is to implement a slightly different function with respect to the original one, provided that the error is acceptable and key parameters are improved.

Approximate solutions are typically obtained by a heuristic procedure that modifies the original implementation. In the case of SW approximation, programmers can typically declare which parts of a program can be computed approximately and specialized compiler and optimizer then preform requested approximations (e.g., EnerJ [4]). In the case of HW approximation, either general-purpose or circuit-specific approximation methods have been applied. While the aim of general-purpose approximation methods (e.g., SALSA [5] and SASIMI [6]) is to automatically approximate any circuit regardless of its structure, circuit-specific methods are focused on a rather specific class of circuits (e.g., adders [7] or multipliers [8]).

III. Evolutionary approximation

The approximation problem can be formulated as a multi-objective optimization problem in which the error, delay (or performance) and power consumption are conflicting design objectives and solved using a general-purpose optimization method. For circuit and low-level code approximations, Cartesian genetic programming (CGP) has provided excellent trade-offs between the key parameters, often better than competitive methods [9], [10].

CGP is a form of genetic programming where each candidate solution is represented as a string of integers of fixed length that is mapped into directed acyclic graph (DAG) [11]. The DAG is constructed within an array consisting of $n_c \times n_r$ programmable n_a-input nodes whose functions are taken from a set Γ. The DAG utilizes n_i primary inputs and n_o primary outputs. No feedback is allowed in the basic version of CGP. The string representation of the DAG is called the chromosome and can be understood as a simplified netlist or assembly language program (Fig. 1).

The search is usually performed using a simple $(1 + \lambda)$ evolutionary algorithm. In this algorithm, every new population consists of the best individual of the previous population and its λ offspring created using a mutation operator which modifies up to h genes (integers) of the chromosome. In order to evaluate the population, each candidate solution is evaluated using the so-called fitness function which assigns better scores to better performing solutions. The search is typically terminated after generating and evaluating a given number of populations.

The CGP-based approximation can be conducted in such a way that CGP tries to firstly modify the exact solution to obtain an approximate solution showing desired value of a given parameter (e.g. the error), and then to optimize the remaining parameters without worsening the parameter optimized in the first step [9]. In a truly multi-objective scenario, all parameters are optimized together [10].

IV. Histogram of Oriented Gradients

The *Histogram of oriented gradients* (HOG) feature extraction is a method widely used for detection of objects such as pedestrians [12]. Pedestrian identification is important in various problem domains, such as surveillance, robotics or driver assistance systems [13]. Although HOG extraction employs the hand-crafted feaures, it is still computationally expensive. For example, for real-time processing of HDTV video stream, a workload of more than 440 Gbps and memory bandwidth of 55 Gbps is necessary [14]. There is a number of works dealing with an efficient implementation of HOG-based detectors (for example, [1], [13], [15]).

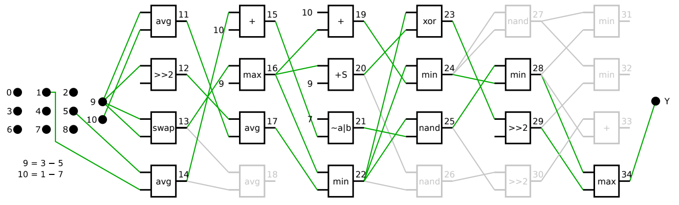


Fig. 1. A candidate solution in CGP (see C code in Fig. 4)

The HOG algorithm can be divided into several steps as shown in Figure 2. The image is scanned by detection windows of selected size. The content of the window is divided into smaller parts and for each of them, the gradient orientation histogram is computed. The histograms are normalized to compensate for different lightning conditions in different parts of the image and the so-called detection window descriptor is created. Finally, the descriptor is fed into a classifier to evaluate whether the window contains specified object or not.

First, the gradient orientation $\theta(x,y)$ and magnitude $m(x,y)$ are calculated for each pixel $f(x,y)$ as follows:

$$\theta(x,y) = \arctan \frac{f_y(x,y)}{f_x(x,y)}, \tag{1}$$

$$m(x,y) = \sqrt{f_x^2(x,y) + f_y^2(x,y)} \tag{2}$$

where $f_x(x,y)$ is the gradient in the x-axis and $f_y(x,y)$ is the gradient in the y-axis, computed as the difference of the adjacent pixels:

$$f_x(x,y) = f(x+1,y) - f(x-1,y), \tag{3}$$

$$f_y(x,y) = f(x,y+1) - f(x,y-1). \tag{4}$$

Next, the image is divided into *cells* of 8×8 pixels and the orientation histograms are calculated for each cell. The interval of 0–180° is evenly divided (usually into 9 parts) to produce the histogram bins. For each pixel, the gradient orientation $\theta(x,y)$ determines the bin and the gradient magnitude $m(x,y)$ is used to calculate the pixel's contribution to that bin. To avoid aliasing, the two nearest bins are updated as well. If the pixel belongs to bin b, the contribution v_b to bin b and contributions to the nearest bins $v_{b\pm1}$ is determined as:

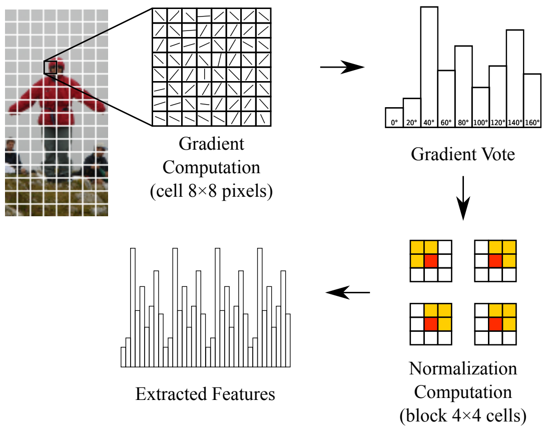
$$v_b = (1-\alpha) \cdot m(x,y), \qquad v_{b\pm1} = \alpha \cdot m(x,y), \tag{5}$$

where α is the weight of the pixel calculated as:

$$\alpha = (b + 0.5) - \frac{n \cdot \theta(x,y)}{\pi} \tag{6}$$

where n denotes the total number of bins (usually 9).

The last step of HOG extraction is normalization of the orientation histograms in order to compensate for lightning



Gradient Computation (cell 8×8 pixels)
0°
20°
40°
60°
80°
100°
120°
140°
160°
Gradient Vote
Normalization Computation (block 4×4 cells)
Extracted Features

Fig. 2. Steps performed to compute histogram of oriented gradients.

and contrast differences in various areas within the image. The cells are organized into overlapping *blocks*. Normally, each block has the size of 2×2 cells, forming a vector of 36 values (considering 9 histogram bins used in cells). The values in each block are normalized using selected scheme and the normalized values of all blocks are then combined in the resulting feature vector. If L2-norm is used, the normalized values can be obtained as follows:

$$v_i^n = \frac{v_i}{\sqrt{\|v\|_2^2 + \varepsilon^2}}, \text{ where} \tag{7}$$

$$\|v\|_2^2 = v_1^2 + v_2^2 + \cdots + v_n^2 \tag{8}$$

and $i = 1 \ldots 36$, v_i and v_i^n denote unnormalized and normalized values of the i-th component of the block vector. ε is a small constant to avoid division by zero.

In the task of pedestrian detection, the input image is scanned using a detection window of 64×128 pixels, which is divided into 7×15 cells (4 pixels on each side serve as a margin) and the resulting feature vectors fed into a classifier contain 3870 values.

V. Proposed approximation

The HOG feature extraction is highly error resilient. For example, Chen et al. [15] replaced various components of HOG with approximate implementations and the classification accuracy remained very close to the standard HOG. However, no systematic approximation approach was employed, the inexact modules were created ad-hoc from standard components such as adders or comparators.

In order to create an approximate implementation of the HOG algorithm, we divided the HOG extractor into several modules, roughly corresponding to the stages of preprocessing (see Fig. 2). There are modules computing the gradient orientation, gradient magnitude, L2-norm, and normalized values.

Then, we performed an error resilience analysis to estimate the contribution of each module to the overall detection performance. The outputs of modules were simply replaced with random values. Table I shows the classification accuracy obtained using 10-fold cross validation on a set of 1836 images taken from the MIT pedestrian dataset [16] and the INRIA person dataset [17]. The most error-resilient modules appears to be those dealing with gradient computation, while a failure in magnitude computation has lower impact on resulting detection accuracy than orientation computation. Note that if both these modules are replaced by a random number generator, the detection is not able to work at all. Similarly, if the L2-norm calculation is broken, the detection accuracy drops to around 50 %, which is the accuracy of a random classifier.

The gradient orientation is computed using the arctan function (see Formula 1), which can be implemented in hardware (or in processors not supporting the arctan) using CORDIC iterative algorithm. In our approach, arctan is replaced by a simple code containing neither loops nor branches. This is useful not only for SW implementations, but also for HW accelerators as the simple code can easily be mapped to a combinational circuit.

TABLE I
Classification accuracy after introducing errors into HOG (10-fold cross validation, 1836 training images).

	Detection accuracy
Fully-functional implementation	99.4 %
Random gradient orientation	94.2 %
Random gradient magnitude	98.5 %
Random gradient orientation and magnitude	49.8 %
Random L2-norm	50.3 %

The target function is evolved using CGP in the same way as CGP was employed to evolve local image filters [18]. Similarly to calculating a new pixel value in image filters, the output of the gradient orientation is based on the neighbour pixels. The 9-neighbourhood centred on currently processed pixel is used as the primary input of the candidate programs, and additionally the gradients of x- and y-axes $f_x(x, y)$ and $f_y(x, y)$ (eq. 3 and 4) are provided (Fig. 1).

The evolved programs will then be used to replace gradient orientation calculation in standard HOG implementation and evaluated in terms of preprocessing performance and classification accuracy.

VI. Results

A. Evolution of gradient orientation module

Our first case study was dedicated to the evolution of approximate arctan function for later usage in HOG feature extraction.

1) Experimental setup: The approximate gradient orientation module was evolved using CGP. The training set consists of windows of 3×3 pixels and the gradients of x- and y-axes $f_x(x, y)$ and $f_y(x, y)$ taken from the Lena image. The golden output is calculated using Formula 1. We used the same setup as for the image filter design described in literature [18], i.e. $n_c = 8$, $n_r = 4$, $n_i = 11$, $n_o = 1$, $n_a = 2$, $\lambda = 7$, and the number of mutations per new individual is $h = 5$. According to [18], Γ contains functions working over 8-bit operands (6 logic functions, 3 shift functions, addition, average, minimum and maximum).

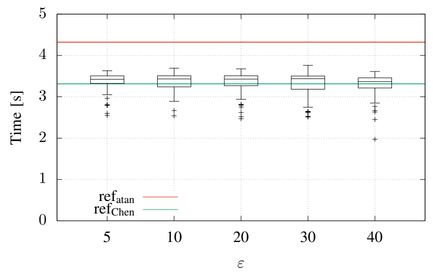
The fitness function is defined as the number of hits:

$$f(s) = \frac{1}{k} \sum_{i=1}^{k} g(v(x, y)), \text{ where} \tag{9}$$

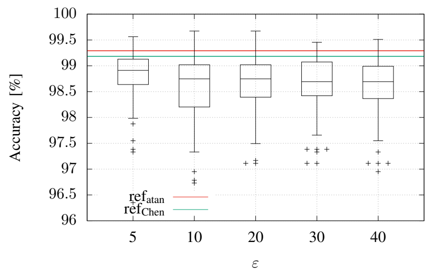
$$g(v(x, y)) = \begin{cases} 0 & |v(x, y) - \theta(x, y)| \geq \varepsilon \\ 1 & |v(x, y) - \theta(x, y)| < \varepsilon \end{cases} \tag{10}$$

where $v(x, y)$ denotes the output of candidate program for pixel at coordinates (x, y), k is the number of fitness cases in the training set, θ is the golden value (see Formula 1), and ε is the allowed error. Five different values of ε are considered in our experiments: 5, 10, 20, 30, and 40.

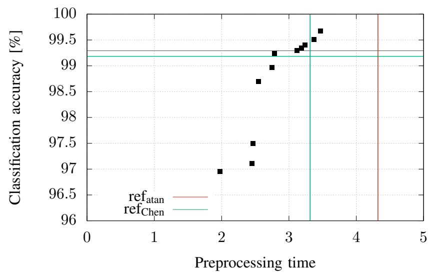
2) Results: We measured the resulting fitness value and the CPU time consumed by CGP from 100 independent runs for each considered ε value. In average, one evolutionary run took 57 minutes of CPU time. As expected, the lowest fitness



(a) Time to process the training dataset for classification.



(b) Classification accuracy.



(c) Pareto frontier: processing time vs. classification accuracy.

Fig. 3. Processing performance and classification accuracy using HOG with various evolved orientation modules.

was achieved with $\varepsilon = 5$ (median value 23.1 %) and as the error boundary relaxes, the achieved fitness is higher; however, none of the evolved solutions fulfill the error boundary for all fitness cases, the highest achieved fitness was 98 % hits (with $\varepsilon = 40$). On the other hand, in the context of HOG extraction, the error of 40° means that the selected histogram bin is at most two bins away from the proper bin, which still results in acceptable accuracy as shown in our following experiments.

B. Replacing orientation calculation by evolved programs

All evolved gradient orientation modules were used as a replacement of the gradient orientation modules in the standard HOG. These implementations were compared with the implementation using standard arctan function (ref$_{\text{atan}}$) and with the implementation described by Chen et al. [15] (ref$_{\text{Chen}}$) in terms of the execution time and classification accuracy. Note that in the ref$_{\text{Chen}}$ approach, the histogram bin is selected using a lookup table, without calculating the exact value of gradient orientation. The remaining parts of the HOG algorithm are implemented in the same way in all compared approaches, the only difference is in the gradient orientation module.

1) Experimental setup: A dataset consisting of 1836 images was used, where 924 positive (i.e. images containing a person) samples were taken from the MIT pedestrian dataset [16] and 912 negative samples (without a person) were randomly chosen from the INRIA person dataset [17]. Each of the compared approaches was employed to extract features from the dataset and a linear SVM classifier was trained using LIBLINEAR [19] to perform the final detection. The evolved programs were translated into a C code and compiled into binary before executing feature extraction to avoid performance bias caused by interpreting the CGP chromosome during runtime. The trained classifiers were evaluated using 10-fold cross validation and compared against each other. The performance of evaluated programs is estimated by measuring the CPU time required to process the whole training set.

2) Classification accuracy and resource usage: Figure 3a shows the CPU time necessary to extract features from the training set obtained for all evolved programs together and for the two reference approaches. Regarding the execution time, all evolved modules beat the ref$_{\text{atan}}$ implementation (with

```
#define SWAP(A, B) (((A & 0x0F) << 4) | ((B & 0x0F)))
#define ADD_SAT(A, B) ((A > 0xFF - B) ? 0xFF : A + B)

unsigned char cgp_atan(unsigned char inputs[11])
{ // all variables are unsigned char.
  n11 = (inputs[9] + inputs[10]) >> 1; n12 = inputs[9] >> 2;
  n13 = SWAP(inputs[9], inputs[9]); n14 = (inputs[5] + inputs[1]) >> 1;
  n15 = n14 + inputs[10]; n16 = MAX(n13, inputs[9]);
  n17 = (n12 + n11) >> 1; n19 = inputs[10] + n16;
  n20 = ADD_SAT(n16, inputs[9]);  n21 = (~inputs[7]) | n15;
  n22 = MIN(n16, n17);  n23 = n22 ^ n20;
  n24 = MIN(n22, n19);  n25 = ~(n22 & n21);
  n28 = MIN(n25, n24);  n29 = n23 >> 2;
  n34 = MAX(n28, n29);
  return n34;
}
```

Fig. 4. The C code of evolved gradient orientation module

the average speedup of 1.28). An average evolved solution shows the same execution time as ref$_{\text{Chen}}$. However, CGP provided many implementations that are much faster than ref$_{\text{Chen}}$ (see the plus symbols in Figure 3a). The experiments were performed on a full-featured Intel Xeon CPU supporting the arctan function. If a simple embedded processor were used, we would expect a more significant difference in performance.

Although the reference implementations show better accuracy than an average evolved solution (Fig. 3b), CGP was able to find superior solutions in both considered objectives. Figure 3c shows the Pareto frontier with four programs outperforming ref$_{\text{Chen}}$ in both objectives, while three of them also outperform ref$_{\text{atan}}$. Having this set of non-dominated solutions, the user can select the right one with respect to constraints imposed by a given application. An example of program from the Pareto frontier is shown in Figure 4. This solution can easily be implemented in SW as well as HW.

VII. Conclusion

We used CGP to evolve new approximate implementations of the arctan function. We integrated these evolved solutions into HOG feature extraction algorithm to detect pedestrians in image data, where the arctan function is typically employed to compute the gradient orientations. We have shown that the best evolved implementations improve not only the execution time, but also the classification accuracy.

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science – LQ1602 and by BUT IGA project FIT-S-17-3994.

REFERENCES

[1] A. Suleiman and V. Sze, “An energy-efficient hardware implementation of HOG-based object detection at 1080 HD 60 fps with multi-scale support,” *Journal of Signal Processing Systems*, vol. 84, no. 3, pp. 325–337, 2016.

[2] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[3] S. Mittal, “A survey of techniques for approximate computing,” *ACM Comput. Surv.*, vol. 48, no. 4, p. 62:162:33, 2016.

[4] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, “EnerJ: Approximate data types for safe and general low-power computation,” in *Proc. of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2011, pp. 164–174.

[5] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, “SALSA: systematic logic synthesis of approximate circuits,” in *The 49th Annual Design Automation Conference*. ACM, 2012, pp. 796–801.

[6] S. Venkataramani, K. Roy, and A. Raghunathan, “Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits,” in *Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1367–1372.

[7] H. Jiang, J. Han, and F. Lombardi, “A comparative review and evaluation of approximate adders,” in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 343–348.

[8] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, “A comparative evaluation of approximate multipliers,” in *IEEE/ACM International Symposium on Nanoscale Architectures*. IEEE, 2016, pp. 191–196.

[9] Z. Vasicek and L. Sekanina, “Evolutionary approach to approximate digital circuits design,” *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[10] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, “Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” in *Proc. of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. EDAA, 2017, pp. 258–261.

[11] J. F. Miller, *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[12] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893.

[13] M. Hemmati, M. Biglari-Abhari, S. Berber, and S. Niar, “HOG feature extractor hardware accelerator for real-time pedestrian detection,” in *2014 17th Euromicro Conference on Digital System Design*, Aug 2014, pp. 543–550.

[14] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, “Architectural study of HOG feature extraction processor for real-time object detection,” in *2012 IEEE Workshop on Signal Processing Systems*, Oct 2012, pp. 197–202.

[15] P. Y. Chen, C. C. Huang, C. Y. Lien, and Y. H. Tsai, “An efficient hardware implementation of HOG feature extraction for human detection,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 656–662, April 2014.

[16] “MIT pedestrian data.” [Online]. Available: http://cbcl.mit.edu/software-datasets/PedestrianData.html

[17] “INRIA person dataset.” [Online]. Available: http://pascal.inrialpes.fr/data/human/

[18] L. Sekanina, S. L. Harding, W. Banzhaf, and T. Kowaliw, *Cartesian Genetic Programming*. Springer, 2011, ch. Image Processing and CGP, pp. 181–215.

[19] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LIBLINEAR: A library for large linear classification,” *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.