Cooperative Coevolutionary Approximation in HOG-based Human Detection Embedded 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 cooperative coevolutionary Cartesian genetic programming (CGP) to exploit the error resilience in the HOG algorithm. We evolved new approximate implementations of the arctan and square root functions, which are typically employed to compute the gradient orientations and magnitudes. When the best evolved approximations are integrated into the software implementation of the HOG algorithm, not only the execution time, but also the classification accuracy was improved in comparison with approximations evolved separately using CGP and also compared to the state-of-the art approximate implementations. As the evolved code does not contain any loops and branches, it is suitable for the follow-up low-power hardware implementation.**

***Index Terms*—Approximate computing, Cartesian genetic programming, Cooperative coevolution, Histogram of oriented gradients**

I. Introduction

Machine learning applications are moving to battery powered and low power embedded devices. Providing high-quality outputs in this context, 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.

One of the key distinguishers of various proposed implementations of machine learning algorithms is the method used to define the so-called features, i.e. the vectors of parameters that represent particular input images, which are used as input for the classifier. Although better classification accuracy can be obtained using *learned features*, which are derived from training data in the process of learning of deep convolutional neural networks, their utilization requires considerable computing resources. On the other hand, the algorithms based on *hand-crafted features* created by experts and used in application-specific algorithms (such as the histogram of oriented gradients, Harr wavelets and shape contexts), can lead to a significant reduction of computation cost and energy in applications, where the lower quality of output (compared to the usage of learned features) is sufficient for that application. For example, the object detection based on deep neural networks led to the 2–4 orders of magnitude overhead in energy consumption with respect to a conventional machine learning system utilizing hand-crafted features as measured on two integrated circuits recently developed for embedded vision [1], [2].

This paper deals with improving of the *histogram of oriented gradients* (HOG) feature extraction method in terms of execution time (which can be translated to energy consumption) in the context of low power pedestrian detection used in driver assistant systems. The error resilience analysis we performed in out previous work [3] revealed that the HOG algorithm is highly error resilient and several components can be considered for approximations with a minor impact on the quality of result, but significant reduction of the execution time and potentially power consumption. 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 [4].

In context of the HOG algorithm, we focus on the gradient histogram computation, which consists of gradient orientation and magnitude computation, both typically based on arctan and square root functions. These are usually implemented by means of relatively expensive iterative algorithms. We use Cartesian genetic programming (CGP), which, in fact, evolves code represented as a directed acyclic graph containing neither branches nor loops, to approximate both the functions. The resulting code can be easily implemented as a program for embedded processor or a combinational circuit for hardware accelerators of HOG. Each function can be evolved separately; however we employ a cooperative coevolutionary algorithm in order to design the functions concurently. We hypothesize that a better combination of approximate implementations can be discovered as a vigorous approximation of one component can be compensated by a gentle approximation of the second one for a certain subset of input vectors that is important in our application. The evolved solutions are 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.

The rest of the paper is organized as follows. Section II introduces relevant previous work. Section III deals with the histograms of oriented gradients method used in low cost image detection systems. The proposed evolutionary approximation approach is presented in Section IV. Experimental results are summarized in Section V. Conclusions are given in Section VI.

978-1-5386-9276-9/18/$31.00

II. Related Work

A. Approximate Computing

The goal of approximate computing is to provide more energy efficient, faster, and less complex computer-based systems by allowing some errors in computations [4]. There is a wide spectrum of approximation techniques, such as precision scaling, loop perforation, task dropping/skipping, memory access skipping, and using of different SW/HW versions. We are primarily interested in functional approximation whose principle is to implement a slightly different function with respect to the original one, but with improved key non-functional parameters and acceptable error.

Approximate solutions are typically obtained by a heuristic procedure that modifies the original implementation. In the case of SW approximation, programmers can declare which parts of a program can be computed approximately and specialized compiler and optimizer then preform requested approximations (e.g., EnerJ [5]). This approach assumes that the processor contains quality-configurable units (such as ALUs, memory subsystem, interconnects) capable of delivering requested energy savings. In the case of hardware approximation, either general-purpose or circuit-specific approximation methods have been developed. While the aim of general-purpose approximation methods (e.g., SALSA [6], SASIMI [7]) is to automatically approximate any circuit regardless of its structure, circuit-specific methods are focused on a rather specific class of circuits (such as adders or multipliers [8]).

B. Evolutionary approximation

The approximation problem can be formulated as a multi-objective optimization problem which can be solved using a general-purpose optimization method. The error, performance (or delay) and power consumption are examples of typical conflicting design objectives. CGP has been successfully used to find high-quality approximate circuits and programs that show excellent tradeoffs between the key design parameters [9], [10].

In CGP, each candidate solution is represented as a directed acyclic graph (DAG), encoded as a fixed-length string of integers called chromosome, mapped into an array of $n_c \times n_r$ programmable n_a-input nodes whose functions are taken from a set Γ [11]. The DAG utilizes n_i primary inputs and n_o primary outputs. As the basic version of CGP forbids feedback connections, the resulting code contains neither branches nor loops. On the other hand, parallel processing and multiple outputs are naturally supported. See example in Fig. 1.

In CGP, a simple (1 + λ) search algorithm is employed, i.e. every new population contains the best individual of the previous population and its λ offspring. In standard CGP, only mutation operator is used, which modifies up to h genes of the chromosome. Each candidate solution is evaluated using the fitness function which assigns better scores to better performing solutions. Typically, the search is terminated after a given number of populations is generated and evaluated.

In order to obtain desired approximations, CGP can be combined with a multi-objective algorithm such as NSGA-II [10]. However, a less computationally expensive but still well perfoming is a preference-driven approach, in which the CGP first modifies the exact solution until the approximation satisfies a given preferred parameter (e.g. the error), followed by optimization of the remaining parameters without worsening the parameter optimized in the first step [9].

C. Cooperative coevolution

The major drawback of the evolutionary approximation is the limited scalability. When dealing with complex programs, the time required for evolution of a candidate program exponentially increases with the number of inputs and size of the program [11]. One possible solution is to divide the problem into smaller components, create their approximations and combine them together to form approximate implementation of the original complex program.

In the coevolutionary algorithms, the fitness of each individual depends not only on its own properties, but also on properties of other individuals [12]. In the cooperative version, typically two or more individuals form a team and their fitness is determined by the overall performance of the team. In the context of complex problem approximation, individuals can represent different components of the problem, but each of them is evolved in a separate population. These populations interact by sending promising candidate solutions to each other, which are then used for fitness evaluation. Several collaboration models have been introduced. In the simplest *1 + 1 collaboration model*, each population is represented by one individual, which is then used for evaluation in other populations. More complex models try to use more individuals from other populations in various ways, for example, the *1 + N collaboration model* or *reference sharing* [13].

There are many applications of cooperative coevolution, for example in function optimization [14], neuro-evolutionary algorithms, where structure and weights of artificial neural networks are coevolved [15], or in evolutionary circuit design [16].

III. Histogram of Oriented Gradients

The histogram of oriented gradients feature extraction is a method widely used for detection of objects such as pedestrians [17]. Pedestrian identification is important in various

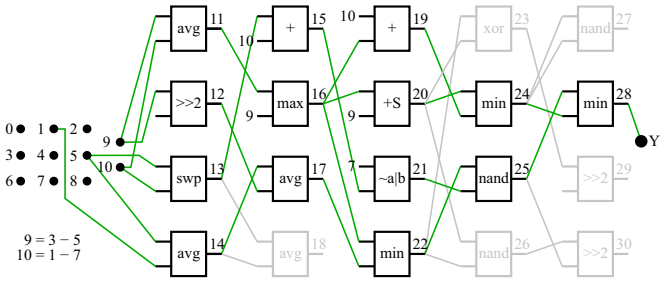


Fig. 1. A candidate solution in CGP

problem domains, such as robotics, surveillance, or driver assistance systems [18]. 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 [19].

The HOG algorithm consists of several steps as shown in Figure 2. First, 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. To compensate for different lightning conditions in different parts of the image, the histograms are normalized. Then the so-called detection window descriptor is created, which 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:

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

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

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:

$$f_x(x,y) = f(x+1,y) - f(x-1,y), \quad (3)$$

$$f_y(x,y) = f(x,y+1) - f(x,y-1). \quad (4)$$

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:

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

where α is the weight of the pixel calculated as:

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

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

In order to compensate for lightning and contrast differences in various areas within the image, the orientation histograms are normalized as the last step of HOG extraction. The cells are organized into overlapping *blocks*. Usually, 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:

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

$$\|v\|_2^2 = v_1^2 + v_2^2 + \cdots + v_n^2 \quad (8)$$

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. To avoid division by zero, a small constant ε is used.

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

There is a number of works dealing with an efficient implementation of HOG-based detectors. The algorithm can be modified in several ways in order to improve performance. Floating point calculation can be replaced by fixed point computations. The inversed square root (eq. 2) can be replaced by look-up table or approximation, so only additions and multiplications are used. Look-up tables can be also used to determine histogram bin, avoiding the need to implement the arctan function (eq. 1) [18]. Additionaly, it is possible to implement some parts of the algorithm in parallel, sacrificing area on a chip for better performance. For example, in Suleiman et al. [1] the usage of multiple parallel detectors allowed for lower power consumption when dealing with image scaled to multiple resolutions.

IV. Proposed Approximation

The HOG feature extraction is highly error resilient. For example, Chen et al. [20] replaced various components of HOG with approximate implementations and the classification accuracy remained very close to the standard HOG. However,

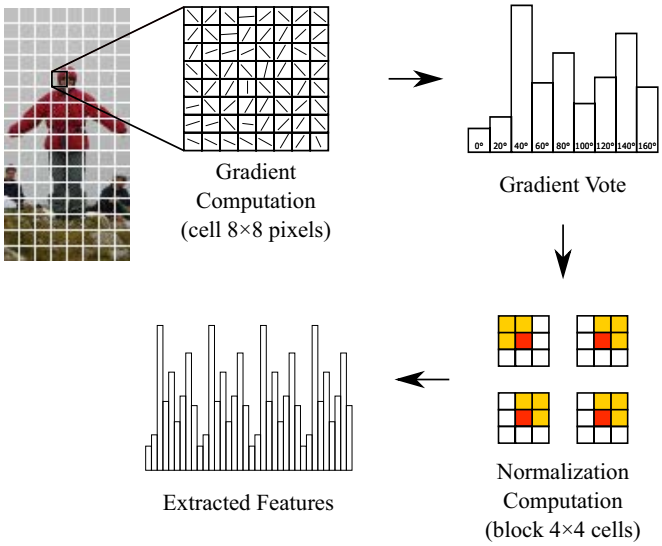


Fig. 2. Steps performed to compute histogram of oriented gradients.

no systematic approximation approach was employed and 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.

In our previous work [3], in which we approximated the gradient orientation module, we performed an error resilience analysis by replacing outputs of modules with random values in order to estimate the contribution of each module to the overall detection performance. We found out that the most error resilient modules are the gradient magnitude computation, followed by the gradient orientation computation.

To further exploit the error resilience in the HOG algorithm, we propose to replace both gradient orientation and magnitude modules by their approximate versions. Instead of the original arctan and square root functions (eq. 1 and 2), which can be implemented in hardware using a relatively expensive CORDIC iterative algorithm, we use a simple code with no loops and branches, which can be easily mapped to a combinational circuit.

These new functions are evolved using CGP in the same way as CGP was employed to evolve local image filters [21]. Similarly to calculating a new pixel value in image filters, the outputs of the gradient orientation and magnitude are 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.

Although both these modules appear to be suitable for replacement by their approximate version, the detection stops working (with accuracy close to a random classifier) when both of them are replaced by a random number generator. To reduce the potential risk of significant drop in the classification accuracy, we evolve the approximate implementations using a cooperative coevolutionary algorithm. We assume that the coevolution should be able to find combinations of orientation and magnitude modules, where weaknesses in one of the modules are compensated in the other module, which would be difficult to obtain if the modules were evolved independently of each other.

In the coevolutionary approach, each module is represented by a different type of individuals, evolved in separate populations. In the first population, approximations of the gradient orientation modules are evolved, while in the other population the gradient magnitude approximate implementations are sought. Figure 3 shows the overall coevolution scheme.

The individuals are not allowed to migrate between populations. The populations affect each other via the fitness functions. To evaluate a candidate gradient magnitude module, selected candidate orientation module (a *trainer*) is required and vice versa. These trainers are stored in *archives* and periodically replaced by top-ranked individuals found during the evolution (i.e., the 1 + 1 collaboration model is used).

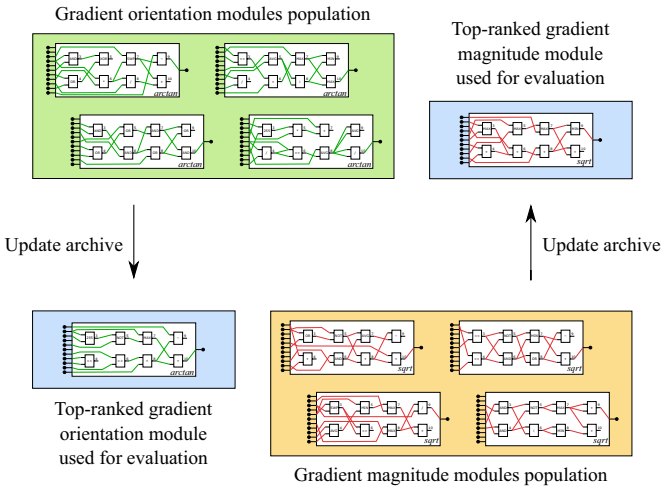


Fig. 3. Schema of the cooperative coevolution approach

The fitness function is based on the comparison of gradient histograms computed using exact HOG algorithm with histograms computed by candidate individual and corresponding trainer. When evaluating the candidate gradient orientation program, the magnitude individual from the archive is used to calculate the gradient histograms and similarly, the archived orientation individual is used during evaluation of the gradient magnitude candidates.

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

V. Experimental results

A. Evolution of gradient orientation and magnitude modules

Our first experiments were dedicated to the evolution of approximate arctan and square root functions for later usage in the HOG feature extraction.

1) Experimental setup: Both modules were 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 (see Figure 5). The modules were evolved separately (cgp_{std}) and using coevolutionary algorithm (cgp_{coe}). The CGP was used in the same way as in the case of local image filters [21], i.e. $n_c = 8$, $n_r = 4$, $l = 1$, $n_i = 11$, $n_o = 1$, $n_a = 2$, $\lambda = 7$, the number of mutations per new individual is $h = 5$ and Γ contains the functions from Table I (all functions are defined over 8 bits).

In the case of cgp_{std}, the evolution was stopped after 300 000 generations. Because only one of the two modules is evolved during one CGP run, it takes 600 000 CGP generations in total to evolve both modules. The coevolutionary algorithm was running in two threads, each dedicated to one population. Each population evolution is stopped after 300 000 generations (the other population can further evolve). The coevolution ends when both the populations reach this limit. The amount of

Fig. 5. Training image for the CGP evolution.

performed work (omitting the coevolution overhead) is thus the same in both cgp_{coe} and cgp_{std}.

To calculate fitness, the gradient orientation and the magnitude calculated for each window in the training set were used to create the gradient histograms (see Formula 5). In cgp_{coe} the candidate program is bundled with a trainer stored in the corresponding archive. In cgp_{std} the exact implementation of module is used in place of the trainer. The calculated histograms were then compared with golden histograms. The sum of absolute differences of values of corresponding bins is used as the fitness function in both populations:

$$f(s) = \sum_{i=0}^{m} \sum_{j=0}^{n} |v_{i_j} - w_{i_j}| \tag{9}$$

where v_{i_j} denotes the j-th bin of gradient histogram of i-th cell computed using the exact algorithm, w_{i_j} denotes the j-th bin of gradient histogram of i-th cell computed using the candidate solution and the trainer (either an individual of the other type or an exact implementation of the other module), m is the total number of cells of 8×8 adjacent central pixels from the windows in the training set and $n = 9$ is the number of bins in histograms.

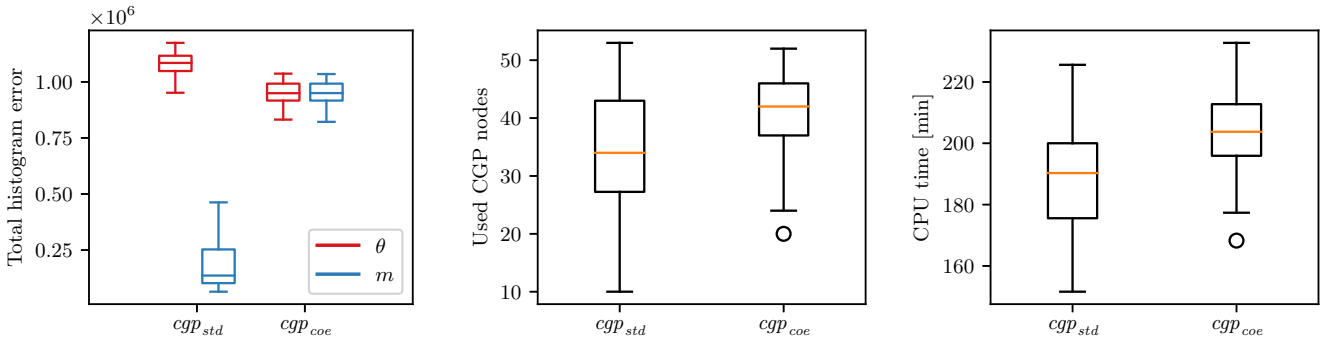
2) Results: We measured the resulting fitness value from 100 independent runs for both coevolutionary cgp_{coe} and non-coevolutionary cgp_{std} approaches. To roughly approximate the area used on a chip (although we did not explicitly optimize for this parameter), we also measured the number of used CGP nodes. The boxplots are shown in Figure 4. For the CPU time, we compare the time necessary to evolve both the gradient orientation and magnitude modules, i.e. the time spent by a single cgp_{coe} run and sum of times spent by two cgp_{std} runs evolving each module individually.

The coevolutionary approach required in average 1.09 times more time to finish, which is caused by the overhead of communication and archives handling, which are not required in the cgp_{std}. Also, the number of active nodes is higher in cgp_{coe}, resulting into more area required on the chip. On the other hand, the coevolutionary approach is able to find better gradient orientation modules in terms of quality. The error of modules found by cgp_{coe} is in average 1.07 times lower than in the case of cgp_{std}. In the case of the gradient magnitude module, lower error is produced by cgp_{std}, but this is mainly due to the fact, that during evaluation the exact gradient orientation module is used and thus each pixel always updates the correct histogram bin. If used with approximate orientation module, the overall error would be higher, as shown in our following experiments.

B. Replacing gradient calculation by evolved programs

In order to evaluate the impact of the quality and performance of the approximate implementations on the HOG algorithm, we replaced the gradient modules in the standard HOG with the evolved modules.

1) Experimental setup: In the case of co-evolved modules (cgp_{coe}), the pairs evolved in the same run were used together. In the case of modules evolved separately (cgp_{std}), there were 100 gradient orientation and 100 gradient magnitude modules and thus 10 000 possible combinations in total. To reduce the number of evaluated combinations, the modules of each type were sorted according to their fitness. Then, the best orientation module was paired with the best magnitude module, the second best orientation module with the second best magnitude module and so on. In total, we considered 100 pairs evolved by cgp_{coe} and 100 pairs evolved by cgp_{std}.



(a) Fitness (b) Used CGP nodes (c) CPU time

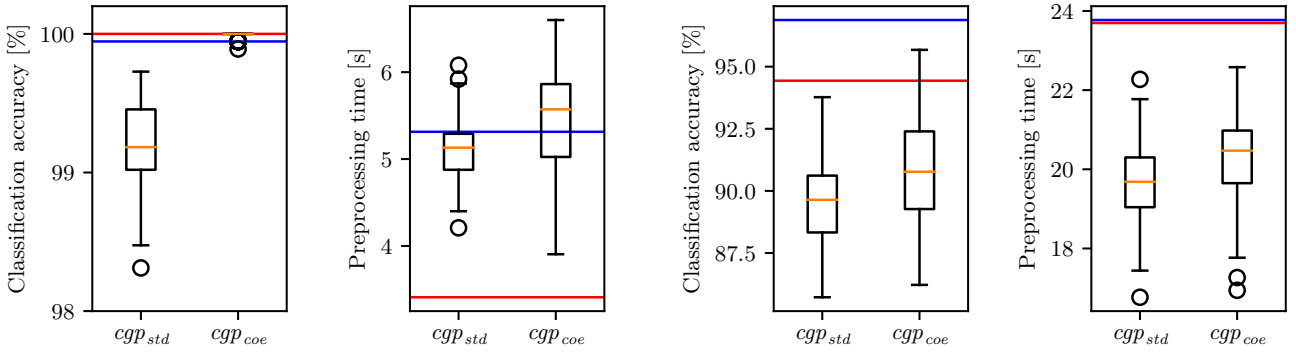
Fig. 4. The resulting fitness and the number of used CGP nodes of the evolved modules, and the CPU time required for evolution of 300 thousand generations. 100 independent runs. Red color (θ) represents the gradient orientation module, blue color (m) the magnitude module.

These implementations were compared with the implementation using standard arctan and square root functions (ref_{exact}) and with the implementation described by Chen et al. [20] (ref_{Chen}) in terms of the execution time and classification accuracy. 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 and gradient magnitude module.

Two datasets were used for evaluation. The first one (*MIT dataset*) consisted of 1836 images. We created it by adopting the whole MIT pedestrian dataset [22], which contains 924 positive images (i.e. images containing a person) and extending it by 912 randomly chosen negative samples (without a person) from the INRIA person dataset [23]. The second dataset (*INRIA dataset*) was sampled from the INRIA person dataset only and it contained in total 4832 images (2416 positive and 2416 negative). The INRIA dataset contains images with persons in various postures and also images of people with bikes, making it more difficult for classification.

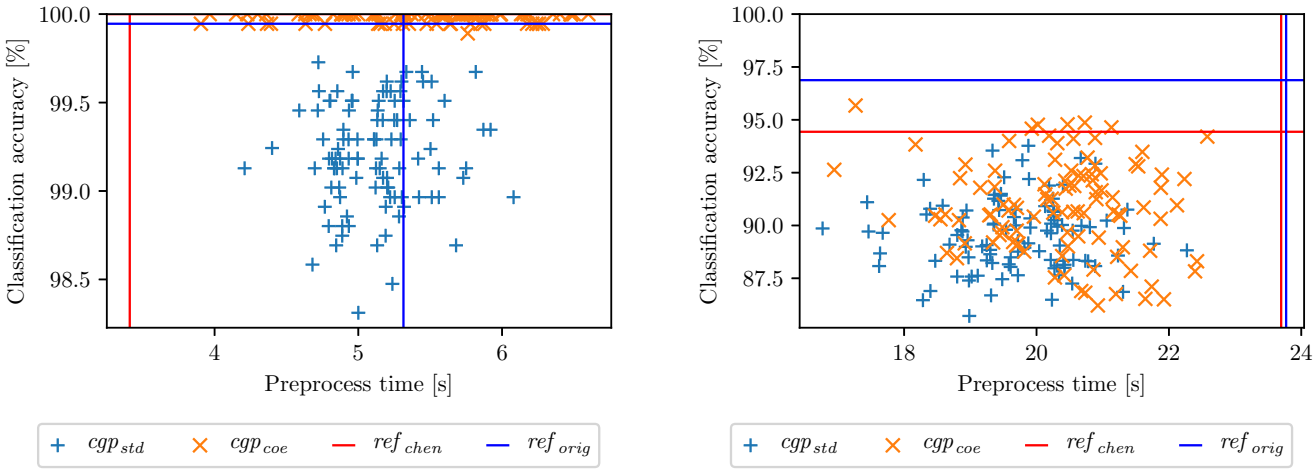
Each of the compared approaches was employed to extract features from the dataset and a linear SVM classifier was trained using LIBLINEAR [24] to perform the final detection. The evolved programs were translated into a C code and compiled into binary before executing feature extraction to avoid performance 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 computational time: Figure 6 shows the CPU time necessary to extract features from the training set and the final classification accuracy obtained for pairs of evolved programs together and for the two reference approaches. In general, the cgp_{coe} results in better accuracy and slightly worse (but still acceptable) processing



(a) MIT dataset (b) INRIA dataset

Fig. 6. The clasification accuracy and time to preprocess the whole dataset after replacing the gradient modules in the HOG algorithm. The blue line is the value obtained using exact HOG implementation, the red line is the value obtained with approximate modules described by Chen et al. [20].



(a) MIT dataset (b) INRIA dataset

Fig. 7. Processing performance and classification accuracy using HOG with various evolved pairs of orientation and magnitude modules.

times compared to cgp_{std}. On the MIT dataset, almost all modules from cgp_{coe} reached 100 % accuracy, while none of the considered combinations from cgp_{std} was able to beat the reference approaches. The processing time is comparable to ref_{exact}.

In the case of the more complex INRIA dataset, the cgp_{coe} has also better resulting accuracy than cgp_{std}, with many implementations better than ref_{Chen}. The required CPU time for preprocessing is better than both reference approaches for all evolved approximate implementations. Note that the experiments were performed on a full-featured Intel Xeon CPU supporting the arctan and square roots function. If a simple embedded processor or hardware implementation was used, we would expect a more significant difference in performance.

Although the reference implementations show better accuracy than an average evolved solution (Fig. 6), coevolutionary CGP was able to find superior solutions in both considered objectives. Figure 7 shows properties of all considered programs. For the INRIA dataset, there are six programs outperforming ref_{Chen} in both objectives. Although none of the programs outperforms $\text{ref}_{\text{exact}}$ in terms of accuracy, all programs reduced the processing time. In the use-cases where a small loss in accuracy can be tolerated, the evolved approximations can be used for speed boost. It is up to the user to select the right program with respect to constraints imposed by a given application. An example of evolved program is shown in Figure 8. This solution can easily be implemented in software as well as hardware.

VI. Conclusion

We used cooperative coevolutionary CGP to evolve new approximate implementations of the arctan and square root functions used in the HOG feature extraction algorithm for pedestrian detection in image data, where these functions are typically employed to compute the gradient histograms.

TABLE I
List of node functions used for gradient orientation and magnitude modules evolution (all functions are defined over 8 bits)

Code	Function	Description
0	255	constant
1	i_1	identity
2	$255 - i_1$	inversion
3	$i_1 \vee i_2$	bitwise OR
4	$\overline{i_1} \vee i_2$	bitwise $\overline{i_1}$ OR i_2
5	$i_1 \wedge i_2$	bitwise AND
6	$\overline{i_1 \wedge i_2}$	bitwise NAND
7	$i_1 \oplus i_2$	bitwise XOR
8	$i_1 \gg 1$	right shift by 1
9	$i_1 \gg 2$	right shift by 2
A	$swap\,(i_1, i_2)$	swap nibbles
B	$i_1 + i_2$	+ (addition)
C	$i_1 +^S i_2$	+ with saturation
D	$(i_1 + i_2) \gg 1$	average
E	$max\,(i_1, i_2)$	maximum
F	$min\,(i_1, i_2)$	minimum

```
#define SWAP(A, B) (((A & 0x0F) << 4) | ((B & 0x0F)))
#define ADD_SAT(A, B) ((A > 0xFF - B) ? 0xFF : A + B)
#define MIN(A, B) ((A < B) ? A : B)
#define MAX(A, B) ((A > B) ? A : B)
#define AVG(A, B) ((A >> 1) + (B >> 1))

// all variables are unsigned char

unsigned char cgp_orientation(unsigned char inputs[11]) {
  n11 = 255 - inputs[10];
  n12 = inputs[9] >> 1;
  n13 = ADD_SAT(inputs[9], inputs[9]);
  n14 = inputs[0] ^ inputs[0];
  n16 = inputs[3] ^ inputs[9];
  n17 = ADD_SAT(n12, n11);
  n18 = SWAP(n13, n14);
  n19 = ADD_SAT(inputs[8], n16);
  n22 = n18 & n17;
  n23 = MIN(n22, n19);
  n27 = AVG(n23, n23);
  return n27;
}

unsigned char cgp_magnitude(unsigned char inputs[11]) {
  n11 = MAX(inputs[8], inputs[3]);
  n12 = AVG(inputs[6], inputs[5]);
  n13 = MAX(inputs[1], inputs[4]);
  n14 = ADD_SAT(inputs[9], inputs[5]);
  n15 = AVG(n11, n13);
  n16 = MAX(inputs[5], inputs[2]);
  n17 = MAX(n12, inputs[7]);
  n18 = n14 ^ n11;
  n19 = MAX(n17, inputs[8]);
  n20 = inputs[6] & inputs[4];
  n21 = n16 + inputs[9];
  n22 = AVG(n15, n18);
  n23 = ADD_SAT(n22, inputs[10]);
  n24 = MAX(n21, n19);
  n26 = ADD_SAT(n20, inputs[6]);
  n27 = n24 | n23;
  n28 = 255 - n24;
  n29 = ~(n26 & n26);
  n30 = inputs[10] + inputs[10];
  n32 = 255 - n27;
  n33 = AVG(n29, n30);
  n34 = n28 + inputs[3];
  n36 = 255 - n34;
  n37 = AVG(n32, n33);
  n40 = MIN(n36, n37);
  return n40;
}
```

Fig. 8. The C code of coevolved gradient orientation and magnitude modules

We have shown that the cooperative coevolution is able to find combinations of approximate functions which together provide better performance in terms of both the execution time and classification accuracy, than combinations of approximate implementations evolved separately. When the evolved implementations are incorporated in the HOG feature extraction algorithm, we have shown that they can improve execution time while the classification accuracy remains comparable with other approximate implementations.

Our future work will be devoted to moving the proposed approach and results to the embedded system and evaluating its performance and energy requirements. Additionally, we will investigate more complex collaboration models in cooperative coevolution in order to increase the classification accuracy while maintaining the improved execution time.

Acknowledgments

The work on this paper was supported by the Czech Science Foundation project 16-17538S and IT4Innovations Excellence in Science project (LQ1602).

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] M. Wiglasz and L. Sekanina, "Evolutionary approximation of gradient orientation module in hog-based human detection system," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Nov 2017, pp. 1300–1304.

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

[5] 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.

[6] 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.

[7] 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.

[8] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 60:1–60:34, Aug. 2017.

[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] E. Popovici, A. Bucci, R. P. Wiegand, and E. D. de Jong, "Coevolutionary principles," in *Handbook of Natural Computing*. Springer, 2011, pp. 988–1028.

[13] M. Shi and S. Gao, "Reference sharing: a new collaboration model for cooperative coevolution," *Journal of Heuristics*, vol. 23, no. 1, pp. 1–30, Feb 2017. [Online]. Available: https://doi.org/10.1007/s10732-016-9322-9

[14] A. Bucci and J. B. Pollack, "On identifying global optima in cooperative coevolution," in *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '05. New York, NY, USA: ACM, 2005, pp. 539–544.

[15] K. O. Stanley and R. P. Miikkulainen, *Efficient evolution of neural networks through complexification*. Citeseer, 2004.

[16] M. Sikulova, G. Komjathy, and L. Sekanina, "Towards compositional coevolution in evolutionary circuit design," in *2014 IEEE International Conference on Evolvable Systems Proceedings*. IEEE, 2014, pp. 157–164.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] "MIT pedestrian data." [Online]. Available: http://cbcl.mit.edu/software-datasets/PedestrianData.html

[23] "INRIA person dataset." [Online]. Available: http://pascal.inrialpes.fr/data/human/

[24] 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.