Evolutionary design of Local Binary Pattern feature shapes for object detection

Filip Kadlček, Otto Fučík
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
{ikadlcek, fucik}@fit.vutbr.cz

Abstract — **This paper deals with the evolutionary design of application specific feature shapes of Local Binary Pattern (LBP) features for object detection in image processing applications. LBP features are very often utilized in image classification systems which are used for pattern recognition. By using genetic algorithm the application of specific weak classifiers' feature shapes, which are highly optimized to achieve a better accuracy of the AdaBoost strong classifier, are being evolved.**

Keywords: Local Binary Pattern (LBP), AdaBoost, Evolutionary designm, feaure shapes.

I. INTRODUCTION

Object recognition and detection are widely exploited techniques in image processing, computer vision applications, security and surveillance systems, artificial intelligence etc. Object detection is usually the first step in the object recognition process and the quality of object detection significantly influences the quality of the whole system.

The classification method AdaBoost [1] is very often used for object detection in image processing. The original AdaBoost method utilizes weak classifiers based on Haar-like features [2], [3], [4]. The later works show that Local Binary Pattern [5] features achieve at least the same accuracy of object recognition as Haar waves. LBP feature processing is easier and faster than the processing of Haar-like features and they are also highly suitable for implementation in Field Programmable Gate Arrays (FPGAs) [6].

The weak classifier, based on LBP, tries to recognize a relatively small piece of input image samples – a texture of classified object. It can be shown that the texture of two different object types may vary widely. LBP features to capture objects' texture depend on shapes, which they utilize. The original LBP shapes are generally defined, so they cannot capture the texture with high precision. If the LBP shapes are tuned according to objects' features, the texture of the object will be captured more precisely. The accuracy of weak classifiers and corresponding strong classifiers is closely related. Another benefit of this approach is reduction of the number of weak classifiers while preserving the same accuracy. It is very important when the strong classifier should be processed in real-time in embedded systems (e.g. using FPGAs).

Recent studies in various areas have demonstrated that evolutionary algorithms (EA) can generate solutions, which have the same or higher quality than conventional solutions in many cases. The basic principle of the evolutionary design is that solutions are encoded as bit strings (chromosomes). Chromosomes are constructed and optimized by the evolutionary algorithm in order to obtain the implementation satisfying the specification given by the designer. The fitness function, which reflects the problem specification, is utilized for quality evaluation of a candidate solution and to ensure convergence to the resultant solution. The fitness function can include behavioural as well as non-behavioural requirements. For example, the correct functionality is a typical behavioural requirement. As a non-behavioural requirement, we can mention the requirement for minimum area occupied on the chip. As the EA is a stochastic algorithm, the quality of the resultant solution is not guaranteed at the end of evolution. However, the method has one important advantage: The artificial evolution can in principle produce (in principle can produce) designs which lie outside the scope of designs achievable by conventional design methods.

This paper deals with the evolutionary design of LBP features shapes. The original feature shapes are designed for a huge amount of classified objects. The main goal of the paper is to demonstrate that weak classifiers based on highly application specific feature shapes, which are evolutionary designed only for classification of one object, achieve better classification accuracy than weak classifiers based on conventional feature shapes. The classification process is computationally intensive, thus a lot of researches create various implementations on various platforms, for example in FPGA [7], [8] or in Graphical Processing Unit [9]. One of the minor goals of this paper is to create optimized feature shapes for application in FPGA.

The remainder of this paper is organized as follows. Section II introduces a LBP feature extraction method, weak classifiers based on LBP and the AdaBoost classifier. In Section III, EA for evolution of feature shapes is introduced. In Section IV, setting of experiments is presented. In Section V, results of EA and classification results are presented. The classification results of EA are compared with the results of conventional approaches. In Section VI, future work is discussed. Finally, conclusions are given in Section VII.

II. CLASSIFICATION BACKGROUND

The classifier is usually used for image pre-processing in complex image processing systems. For the classification problem, a number of various techniques can be used. Because the result of this work is intended for usage in FPGA

technologies, the AdaBoost algorithm in conjunction with LBP (Local Binary Pattern) has been chosen. This combination achieves a very good accuracy in classification [5].

A. Local BinaryPattern

LBP is a non-parametric operator, which was for the first time introduced in [10]. An improvement was introduced later in [11]. LBP is a structural operator, which provides image analysis. The LBP operator tries to recognize an image texture. The basic form of LBP works with a set (matrix) of nine values. In the terms of image processing values are represented by the pixels of an image. The set is created by eight adjoining pixels of a center pixel (the one being processed). In fact it is a *3 x 3* matrix. The evaluation of the LBP operator is done by the following procedure. Each marginal pixel is compared with the centre pixel. If the value of marginal pixel is lower than the one of the center pixel, then the result of comparison is a logical zero; otherwise the result of comparison is a logical one. In the process of evaluating LBP, the comparison is made eight times - eight 1-bit values are obtained. These values are composed to one byte. The composition is predefined. The LBP operator has two hundred and fifty six different output results. The whole process of evaluation of LBP feature is shown in Figure 1.

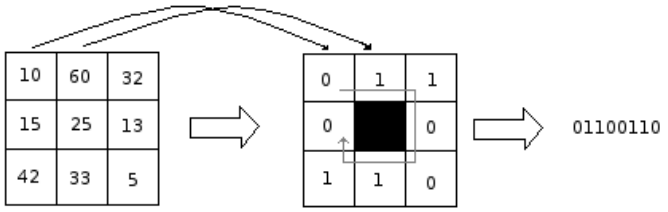

Figure 1: Evaluation of LBP feature

The evaluation of an LBP feature is expressed by the following equation [12]:

$$LBP(x_c, y_c) = \sum_{n=0}^{7} 2^n s(i_n - y_c) \qquad (1)$$

Where y_c is the centre pixel, x_c is a part of the input image, i_n corresponds to the values of 8 neighbourhood pixels and $s(x)$ is a function expressed as:

$$s(x) = \begin{cases} 1, \text{if } x \geq 0 \\ 0, \text{if } x < 0 \end{cases} \qquad (2)$$

LBP is a very powerful operator, which has many benefits. Some examples of them: LBP is an illumination invariant, LBP uses only simple mathematical operators in the evaluation process and LBP is very suitable for FPGA implementation.

B. Multi-scale Block Local Binary Pattern

The LBP operator has good classification accuracy. However, the LBP operator was improved to Multi-scale Block LBP (MB-LBP) [5]. The improvement in MB-LBP is in replacing the basic fields of LBP. The original fields in LBP consist only of one pixel. The original matrix of size *3 x 3* in LBP consists of nine fields. Each field in MB-LBP represents a group of image pixels (data). The field is a continuous area, which has an obvious rectangular shape. The representing value of each field is computed as a convolution of all pixels. The evaluation of MB-LBP is very similar to LBP. The difference is only in the coefficients x_c and y_c in the equation (1). When calculating MB-LBP, x_c represents the convolution of each adjoining field and y_c represents the convolution of pixels under the center field. Several MB-LBPs feature shapes are shown in Figure 2.

The MB-LBP is more robust than original LBP. LBP covers only a small area of the image therefore it may only describe a small piece of the image (*3x3* pixels), but MB-LBP may cover a whole image sample, so it may describe the whole image. MB-LBP can recognize the texture of the image more precisely than LBP.

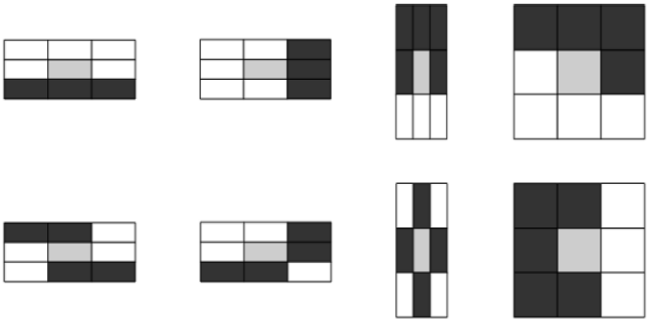

Figure 2: Types of MB-LBP feature shapes [13]

C. Weak classifier

In the preceding paragraphs, the LBP operator and its extensions were described. Let the LBP operator be a function $f_{LBP}(x, y, I)$, where x and y are the coordinates of the left upper corner of the *3 x 3* matrix for LBP in an input image sample *I*. The position of each feature is defined by the coordinates. The function f_{LBP} may return one of 256 different values. The weak classifier is an extension of this function. The weak classifier transforms output the value of the f_{LBP} function to a new value, which represents the probability that a detected object (a positive result of classification) is in the input image sample *I*. The weak classifier (sometimes called weak hypothesis) is represented by a function *h (I)*. In the function *h,* constants *x* and *y* are defined. Constants define the position of the f_{LBP} function in an image sample and there is also a probability table defined which has 256 values (one value for each result of the f_{LBP} function). The probability table is obtained by the training process. Each weak classifier has to be trained first, before it is used for the first time. The training algorithm (in its simple form) of a weak classifier is described below.

Let $S=\{(x_1,y,w_1),(x_2,y_2,w_2),\ldots,(x_N,y_N,w_N)\}$ be a training set, where x_i is an input image sample, $y_i \in \{-1,1\}$ (*-1* – negative sample, *1* – positive sample) and $w_i \in R$ is a weight of the image.
Init T_{HIST} (histogram table) to zero.
Foreach s in S:
$T[f_{LBP}(x,y,s.x)]$ += s.y . s.w;

The function of a weak classifier *h (I)* is described by the code:

```
h(I){
        const x;
        const y;
        return T_HIST[ f_LBP (x,y,I) ]
}
```

Where the constants x, y and the table of constants T_{HIST} are obtained from the training process.

D. AdaBoost

The method called AdaBoost is a statistical method, which is widely used for object detection. This method is an extension of the original method called Boosting [14]. Boosting is a learning method. It is based on statistical combination of many weak hypotheses. AdaBoost [1] extends Boosting by adding weights to samples of a training set. The AdaBoost method consists of two basic parts – a training phase and a runtime phase. The training phase is a machine learning procedure, which was introduced in detail in [1]. The training process is complicated and it is out of the scope of this paper. The second part of the AdaBoost algorithm is the runtime phase. In this phase, the results of training are processed. The process of the runtime phase is described by the following equation:

$$H(X) = \begin{cases} 1, \text{ if } \left(\sum_{i=1}^{N} \alpha_i [h_i(X)] \right) > t \\ 0, \text{ otherwise} \end{cases} \quad (3)$$

The result of the function $H(X)$, for an input image X, is computed as a linear combination of N weak hypothesis results. The linear combination is substituted by a simple sum in this case. The result of this linear combination is compared to the threshold t. If the comparison is positive, then the result of $H(X)$ will be *1* otherwise it will be set to *0* (*1* indicates that the object is detected). The function $h_i(X)$ represents a weak hypothesis (for example based on LBP). The α_i is a constant obtained during the learning process. This constant represents the fidelity of the weak classifier.

III. Evolutionary Design of New Shapes

A novel technique for designing new shapes of LBP features is proposed in this section. A new feature shape is considered as a set of nine convolutions. The inputs of the convolutions are pixels of a grayscale input image sample. The new shapes are created by EA without acceleration in FPGA. The design process will have to be repeated when the application requirements are changed.

A. New geometric shape of features

The original size of LBP features is restricted to *3 x 3* pixels of a source image. The size of MB-LBP features is restricted only by the size of the source image (precisely the size of the detection window). In face detection for example, the maximum size of MB-LBP features may be approximately *24 x 24* pixels. The increasing size of MB-LBP does not produce much better results. For most applications the size of MB-LBP features can be restricted to *6 x 6* pixels. The features with unrestricted size can be difficult to implement into embedded systems using FPGAs as well as to design by EA.

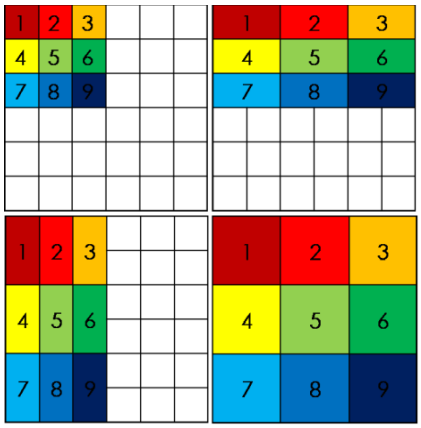

Figure 3: Standard feature shapes of MB-LBP

The standard feature shapes are shown in Figure 3. Each of those four basic shapes is formed by a continuous area. Each field in the feature is also a continuous area and each area has also the same size (for one feature shape). There are no overlapping fields. The original features have to comply with several criterions. The new features designed by genetic algorithm have to comply with one criterion only. The criterion is the maximum size of the area, on which the new shape lies. An example of a possible feature shape is shown in Figure 4.

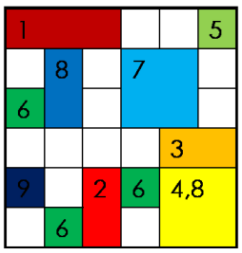

Figure 4: New feature shape example

Figure 4 shows overlapping – fields *4* and *8*, discontinuity – field *6*, different area size –fields *5* and *7*, and discontinuity of the whole feature.

The evaluation of a new shape is similar to evaluation of the original LBP (equation 1). The number of fields in a new designed feature corresponds to the number of fields in the original LBP. Because of different dynamic ranges of all convolutions, normalization should be done to obtain a result as an *8*-bit value.

B. Problem encoding

The candidate solution is needed to be encoded into a chromosome, which is represented by a bit string. The nine fields of the feature shape need to be encoded. Each convolution is encoded by a *36*-bit string. Each bit in a sub-chromosome corresponds to one pixel of a *6 x 6* pixel matrix. If the bit has the value *1*, a corresponding pixel is contained in the convolution. Thanks to the use of a fixed length of a (sub-) chromosome, the operation of mutation and crossover are not difficult and the arbitrary operation returns a valid chromosome (each chromosome is valid). The final chromosome is obtained by merging nine sub-chromosomes. The length of the whole

chromosome is *9 x 6 x 6 = 324* bits. Figure 5shows how the chromosome is created.

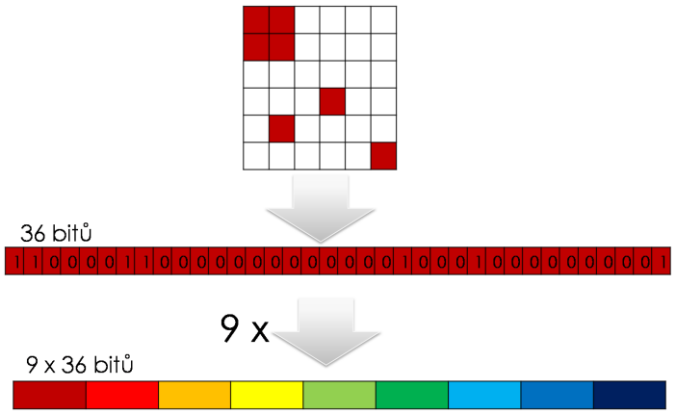

Figure 5: Chromosome creation

C. Genetic algorithm

The classical genetic algorithm GA [15] is used in this work. The GA has two interesting parts – initialization and evaluation of the population by the fitness function, which will be discussed here. There are two basic approaches to the initialization process. The first one is to carry out a guided initialization. In this case the population is initialized by existing shapes. The second approach is to carry out a random initialization. The disadvantage of guided initialization is the risk that the genetic algorithm may be stuck for a long time at a local extreme and it would not achieve the global maximum. GA is using tournament selection, which is making one point crossover and one mutation with predefined probability per chromosome. Due to predefined probability some chromosomes are unchanged after those operations. The creation of a new population is carried out by a technique called elitism (the best chromosomes are automatically inserted to the new population).

D. Fitness functions

The fitness function is the most important part of genetic algorithms. The final solution of GA is the one most affected by the fitness function. The form of fitness function determines how the final shape will work (what accuracy will be achieved). Fitness function for creating shapes, which could achieve the best result only at one place, or shapes which will achieve quite good results at several places, can be established. Also a fitness function, which optimizes the shape for FPGA implementation, can be founded.

In the proposed method, the computation of fitness function is based on weak classifiers training. During the fitness function evaluation a set *C* of weak classifiers is synthesized. All of them are very similar, with the only difference in the position of the feature in the detection window. Each classifier of set *C* is based on the same shape and trained and evaluated separately. The accuracy is established during the training process on a testing data set. Each shape is trained in many ways and there are many results of classification (set *C*). The procedures of obtaining the final fitness function are described in the following subsections.

1) Global maximum

The first choice, how to compute the final value of the fitness function, is based on finding the maximum in set *C* (set of all classifiers' accuracy). The fitness function finds the classifier from set *C*, which has the best accuracy. This accuracy is subsequently propagated as a value of the fitness function. This type of fitness is focused on finding shapes, which have the best accuracy only at one place in the detection window. The accuracy of weak classifiers based on such a shape is around *70 – 75 %*. This fitness optimizes the feature shape during the evolutionary process only for one place. This feature has a very good accuracy, but a strong classifier may contain only a few weak classifiers, which are based on this feature shape (see Section VI).

2) Average and mean value

The next fitness is based on computation of an average or a mean value from the accuracy values of all weak classifiers in set *C*. Those functions find the feature shapes, which are optimal for the whole detection window. It is clear, that at some positions the feature will carry out a good accuracy and at another position it will carry out a poor accuracy. The number of positions, where the good accuracy is achieved, is higher than in the previous fitness. The accuracy achieved by this fitness is around *60 %*. This accuracy is better than the accuracy achieved by the classifiers based on classical feature shapes. Unlike the shapes designed by the Global maximum fitness function, it is possible to construct the whole strong classifier from these weak classifiers.

3) Averege and mean value of N best positions

This type of fitness is based on an average or a mean value of classifier accuracy like the previous fitness. But the difference is in finding the average (or mean) value. In this case, fitness is not computed from the whole set *C* of the classifier but only from *N* best positions. The *N* may be chosen from *10* to *20*. If a bigger *N* is chosen, the results will be very similar to the preceding function. The accuracy of this approach is around *65 %*. It is not possible to synthesize the whole classifier from weak classifiers, which are based on this approach. But in this case, more weak classifiers can be used than in the Global maximum case.

4) FPGA criterium

The last type of a simple fitness function is focused on evaluating the feature shapes according to FPGA resources (slices – logical cells; BRAM – Block Random Access Memories). The numbers of operations needed for the evaluation of each shape can be determined. The set *O* of all possible operators, which are used for computation of LBP, include addition, division and division by power of two.

For each operator is established an implementation cost in FPGA (amount of resources). The total cost of a feature shape is expressed by the following equation:

$$FPGA_{cost} = \sum_{o \in O} f_{cost}(o) \cdot f_{freq}(o) \quad (3)$$

Where f_{cost} is a cost function of the operators; f_{freq} is a frequency function of the operators. Both functions have an input parameter o – an operator. Each shape is evaluated by equation 3. This fitness cannot be used individually, because the result of the evolution algorithm would make no sense from a classification point of view (see next subsection).

5) Combination of fitness functions

Because the FPGA fitness function does not deal with classification accuracy, a combination of fitness functions should be defined. One possible case of a combined fitness function can be expressed by the following equation:

$$F_{multi}(s) = f_{FPGA}(s) \cdot W_{FPGA} + f_{glob\max}(s) \cdot W_{glob\max} \quad (4)$$

Where $f_{FPGA}()$ is the FPGA fitness function; $f_{glob\max}()$ is the global maximum fitness function; s is the shape feature; W_{FPGA} is the weight of the FPGA fitness function; $W_{glob\max}$ is the weight of the global maximum fitness function. The resulting fitness can be used to design the features with good accuracy and suitable for implementation in FPGA.

IV. Experiments

The dataset used in our experiments is composed of two main parts – a training dataset and a testing dataset. The UIUC dataset was used [16]. The training dataset is composed of *1050* image samples. There are *550* positive samples and *500* negative samples. This sample set was used for training strong classifiers and also for training weak classifiers by the evolutionary algorithm. Each training sample has been resized to a width of *40* pixels and a height of *16* pixels. The size of the detection window was also *40x16* pixels. The training set is also used in a fitness function to find out the accuracy of the weak classifier.

The testing set was used to test the accuracy of the strong classifier. The testing set is divided into two different parts. The first part of the training set is composed of *169* images, the objects in the image samples have similar size and one image sample may contain several objects (vehicles in this case). The second dataset is composed of *107* samples and the size of the objects differs. The accuracy of the strong classifier and the results of GA are evaluated on these two datasets. In the experiments, the results of the strong classifier, which utilizes evolutionary designed feature shapes, are compared with the classifier that utilizes only standard shapes.

V. Results

The proposed method was examined in two ways. The first was focused on results of the genetic algorithm; the second one was focused on the results of classification. The fitness function *Average of N best positions* was used in all experiments where *N* was set to *10*.

A. Results of the genetic algorithm

This paragraph examines the parameters of the genetic algorithm. All experiment results are obtained as an average value of three runs.

1) Intialization

Two ways of initialization were examined. The first case is a guided initialization. In the guided initialization, the whole population is initialized by standard shapes. The second case is random initialization. The chromosome is initialized randomly with a given rate. A higher rate determines that the shape will contain more image pixels. The test showed that random initialization with a small rate is the best solution. When we use guided initialization, GA often gets stuck at a local extreme. The results of initialization are shown in Figure 6. The vertical axis expresses the value of the fitness function (the smaller is better); the horizontal axis expresses the number of generations.

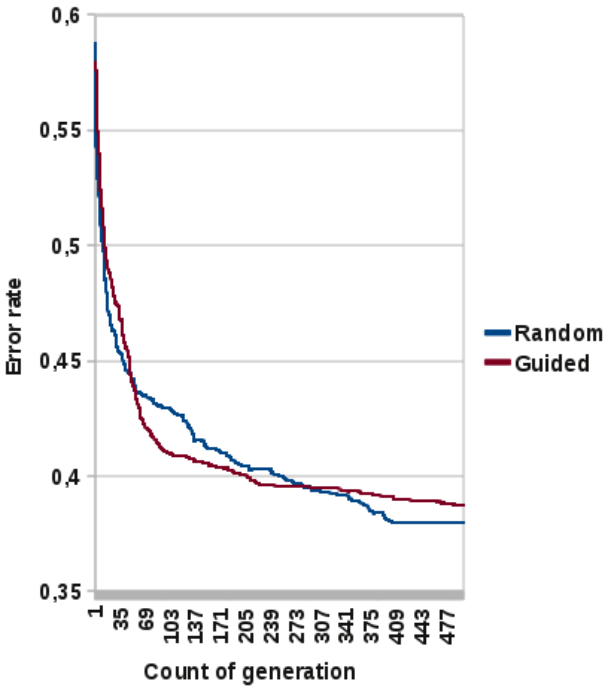

Figure 6: Average Error rate for different initialization of GA

2) Genetic algorithm parameters

The results of experiments with input GA parameters are described in this section. The main goal of this paragraph is to prove that the proposed GA, with good input parameters, converges in most cases. The examined parameters are the number of mutations in a chromosome in one step, mutation rate, population size, and crossover rate. One point crossover was used in our experiments. The parameters of the GA were examined with the goal value of the ending condition set to *500* generations. When the algorithm reaches this number of generations, the last result is taken as the result of the fitness function. The meaning of all axes in the figures is the same as in Figure 6.

Figure 7 shows the results of the experiment, in which the number of mutations was examined. The number of mutations was examined in a range of *0 – 30*. When only crossover (*0* mutations) was used, the GA did not converge to the goal value. In the experiment the following settings were used: mutation rate = *40 %*, crossover rate = *80 %* and population

size = *40*. The best results were achieved, when we used *1* mutation per chromosome.

The next experiment was focused on crossover rate. Figure 8 shows, that the parameter of the crossover rate is less important than the number of mutations, because in most cases satisfactory results were achieved. The best results were achieved when the crossover rate was set to *0 %* and *80 %*. If

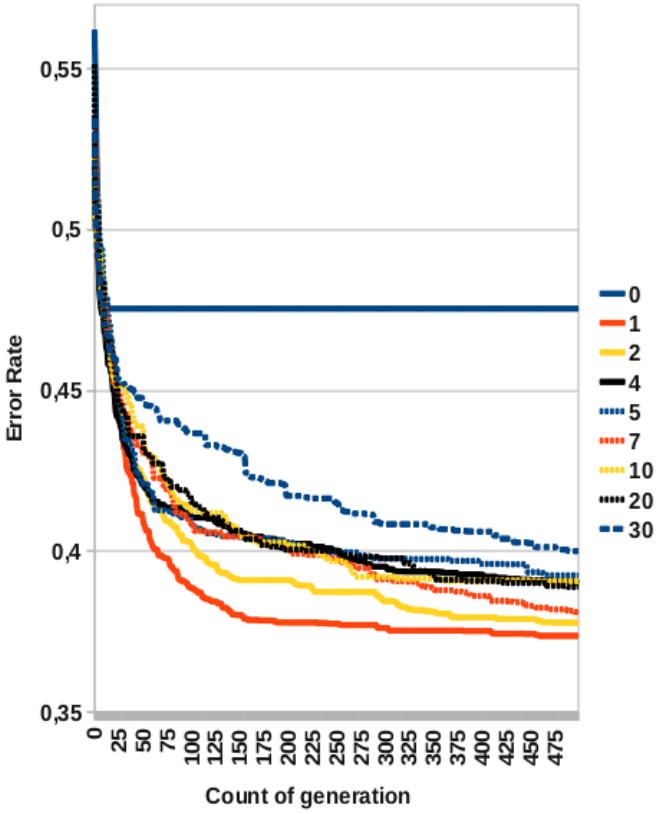

Figure 7: Average Error rate for different count of mutation

the crossover rate is set to *0 %*, the GA often strands in some local extreme. The best crossover rate was selected as *80 %*. The parameters were set to the following values: mutation rate = *40 %*, number of mutations = *1* and population size = *40*.

The last examined parameter of the GA was mutation rate. The results of these experiments are shown in Figure 9. There is also shown, that for the rate of *0 %* the algorithm does not converge (correspondence to the number of the *0* mutations experiment). The best result was achieved for mutation rate = *70 %*. The experiment parameters were set to the following values: crossover rate = *80 %*, number of mutations = *1* and population size = *40*.

The best parameters of the GA from these three results were as follows: mutation rate = *70 %*, crossover rate = *80 %*, number of mutations = 1, one point crossover. The mutation rate and crossover rate are slightly high. But this is caused by a required high variability of the population; the solution space is scanned sparsely but faster (evaluation of a chromosome takes a long time). This is possible because the optimal solution does not need to be necessarily found by EA. With these parameters

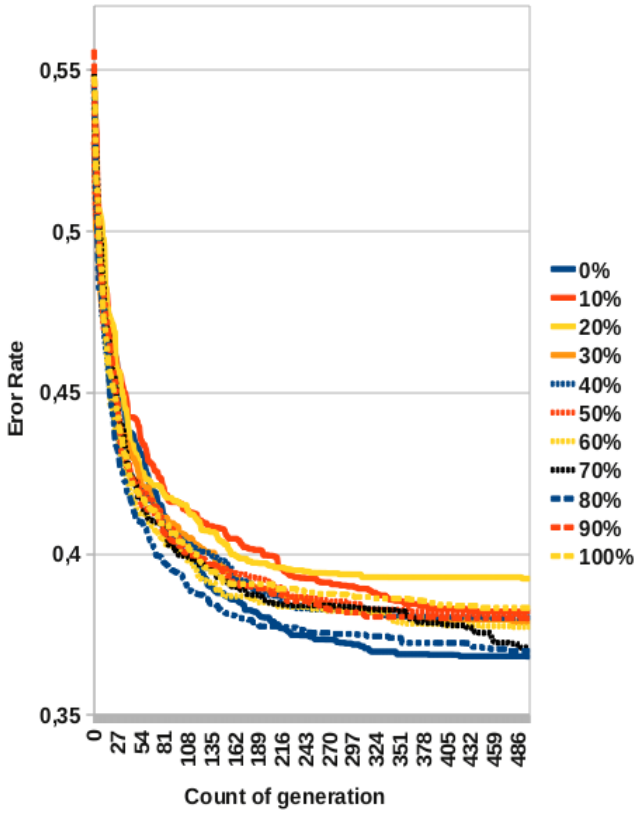

Figure 8: Average Error rate for different crossover rates

faster convergence and smaller time consumption is achieved. One parameter was not yet examined – the population size. In all cases we use a population size = *40*.

3) Convergence of the GA

Obviously the GA iterates until a solution is found. In the proposed method, a solution is found when the required accuracy is achieved (it is not necessarily the best solution). It was examined that the suitable size of the population is around *40* individuals. When a smaller number of individuals is used, the GA does not converge in many cases. When a number that is too big is used, the GA converges in *95 %* percent of cases, but the number of individuals' evaluation is too high and the time needed for computation is too high.

Figure 10 shows the convergence results of the proposed GA. The GA was run *15* times with same parameters and in each run the number of individuals' evaluation was measured, which was represented by an evaluation of the fitness function. In the experiments, the parameters were set to the following values: population size = *40*, mutation rate = *70 %*, crossover rate = *80 %*, one point crossover and number of mutations = *1*.

It is possible to see that nearly all runs have the number of evaluations close to the value of *10 000*. Only one experiment differed. But even so, all experiments ended with a satisfying number of individuals' evaluations.

B. Classification results

In this paragraph, the results of the strong classifier, which are based on the newly designed feature shapes, are examined.

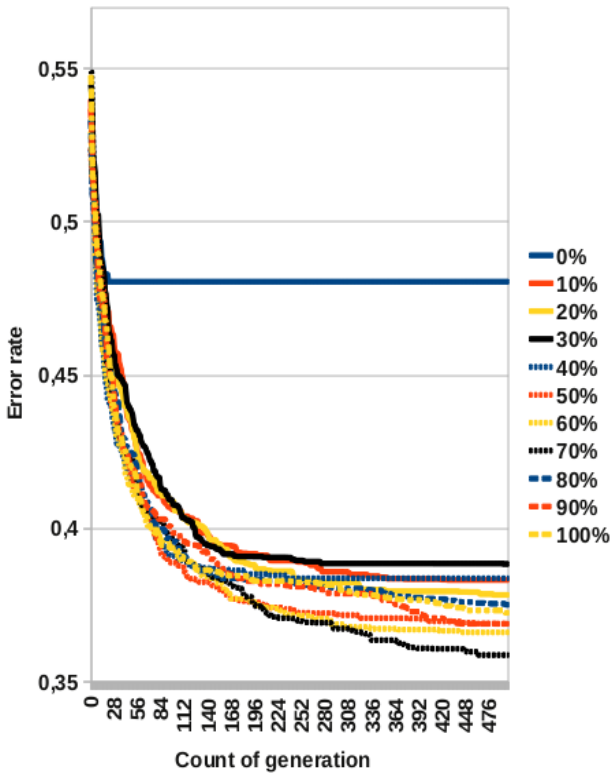

Figure 9: Average Error rate for different mutation rate

The whole strong classifier cannot be composed only of evolutionary designed features, because it would not work (see section VI). Evolutionary based weak classifiers were used only at the start of the classification cascade. However the experiments also show some results with weak classifiers, which were used at the end of the classification cascade. These results are not better than the results obtained by the standard classifier.

Figure 11 shows the results of the strong classifiers. There are results of the classifier based only on standard shapes (*Standard*), and results based on evolutionary designed shapes in conjunction with standard shapes (*EVO X-Y*). In Figure 11, it is possible to see that the evolutionary shapes provide best accuracy when used at the beginning of the strong classifier. The mark *EVO 10-5* means that the strong classifier is composed of three parts. The whole classifier has *100* weak features. The number *10* indicates that the first *10* weak classifiers are based on evolutionary designed shapes. The number *5* indicates that at the end of the strong classifier *5* weak classifiers based on evolutionary design were used. *85* standard weak classifiers are used in this case. The best result is obtained when the classifier *EVO 10 - 0* is used. When we use the following configuration *EVO 1 – 10* then the result of the classification is worse than the classifier based on standard feature shapes. Figure 11 has three parts: Correct Detection, False Detection and F Measure. Correct detection is the rate of correctly detected objects. False Detection is the rate of incorrectly detected images. F. Measure is a combination of the two previously defined indicators.

The newly designed feature shapes are appropriately robust and general. The shapes, which are trained for the application of recognition of one object type, may be used for recognition of another object type. The accuracy of these feature shapes will moderately decrease below the level of standard feature shapes accuracy, however they are still applicable.

VI. Discussion

In this paper, the process of designing new feature shapes and training of strong classifiers are described separately. This division has several disadvantages. The source of the problems is in the weights of image samples, which are used during the classifier training process. These weights are changed during the training process. But in the process of evolutionary design of new feature shapes, the algorithm works with the initial weights all the time. These initial weights are the same for all image samples.

Therefore the evolutionary designed shapes are trained only for the usage at the start of the classification cascade of the strong classifier (AdaBoost). Their application at any position of the classification cascade does not produce satisfying results. This is also shown in Figure 11 (*EVO 1 - 10*). However this problem has a solution. Evolutionary design of feature shapes can be carried out during each training step of the strong classifier (after training one weak classifier). An obstacle of this approach is that the evolutionary design takes a long time and training such classifier will be much longer (several weeks). Another possibility is to carry out evolutionary design at each *N* step of the training process, where *N* may be set from *1* to *10*. By this way training time may be reduced.

This work takes into account only weak classifiers based on the LBP function. There are other weak classifiers, which are very similar to LBP - for example, the Local Rank Pattern [17] and Local Rank Difference [18]. These features work with the same shapes as LBP and they can be also used.

VII. Conclusion

This paper introduces a novel method for design of new weak classifier application specific feature shapes by the genetic algorithm. The proposed method increases the accuracy of AdaBoost classifiers by about *3 – 4 %* (see Figure 11). This is very good improvement, which may also be utilized in different ways. The strong classifier, which utilizes the new feature shapes, may be composed of less weak classifiers to achieve the same accuracy as the original classifier. This is very important when the classifier is implemented in an FPGA. Figure 6– 9 show the results of setting GA parameters. The best parameters were found (see section V.A.2) and verified (see Figure 10 and section V.A.3).

Several fitness functions, which are able to affect the application of shapes, are introduced in the paper. Also a fitness function, which optimizes new shapes for use in FPGA, was introduced. A combination of fitness functions was introduced as an instrument to combine several fitness functions into one. Some feature work was mentioned in section VI. The newly designed feature shapes are intended to be used in the architecture, which was introduced in [19] and where the hardware implementation was done.

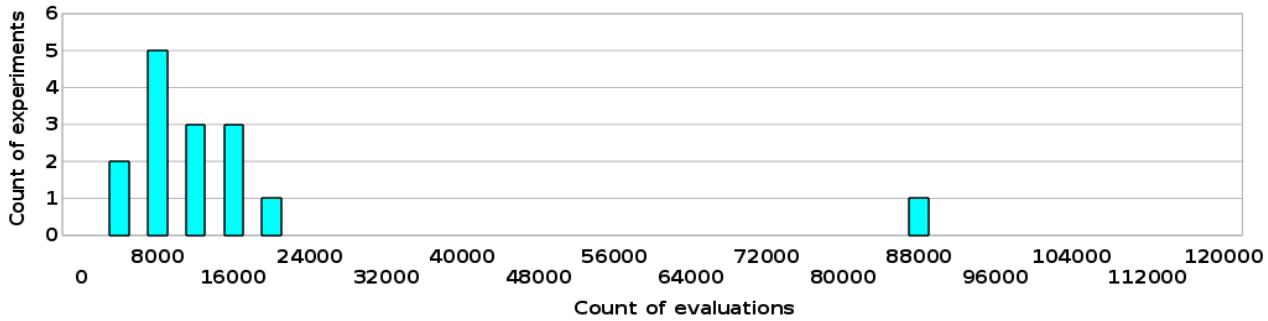

Figure 10: Count of experiments histogram in dependence on count of evaluation

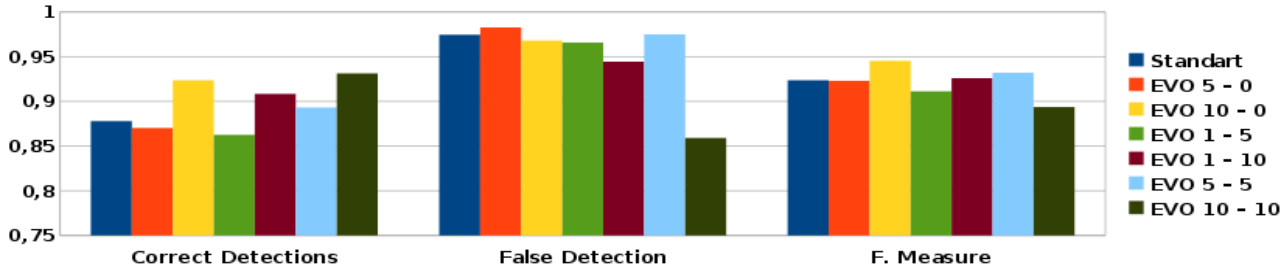

Figure 11: Accuracy of classifiers

ACKNOWLEDGMENT

This paper has been elaborated in the framework of the IT4Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 supported by Operational Programme 'Research and Development for Innovations' funded by Structural Funds of the European Union and state budget of the Czech Republic.

REFERENCES

[1] Viola, P., Jones, M.: Real-time object detection. Int. J. Comput. Vision, vol. 57, no. 2, pp. 137–154, May 2004.

[2] Viola, P. and Jones, M.: Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[3] Haar, A.*:* Zur Theorie der orthogonalen Funktionensysteme, chapter Mathematische Annalen. 1910, s. 331-371.

[4] Wilson, P. I., Fernandez, J.: Facial feature detection using Haar classifiers. *J. Comput. Small Coll.*, ročník 21, č. 4, 2006: s. 127-133, ISSN 1937-4771.

[5] Liao, S., Zhu, X., Lei, Z., et al.: *Learning Multi-scale Block Local Binary Patterns for Face Recognition*. Chinese Academy of Sciences, China, 2007, s. 828-837.

[6] Hradis, M., Herout, A., Zemcik, P.: Local Rank Patterns –Novel Features for Rapid Object Detection. In: Proceedings of *International Conference on Computer Vision and Graphics 2008*, Heidelberg, DE, Springer, 2008, s. 1-12, ISSN 0302-9743.

[7] Kyrkou, CH. A Flexible Parallel Hardware Architecture for AdaBoost-Based Real-Time Object Detection. *IEEE Transactions on very large scale integration (VLSI) systems*, p. 14.

[8] Zemčík, P., Žádnik, M.: AdaBoost Engine. In *Field Programmable Logic and Applications*, Aug. 2007, p. 656-660.

[9] Polok, L., Herout, A., Zemčík, P., Hradiš, M., Juránek, R., Jošth, R.:"Local Rank Differences" Image Feature Implemented on GPU. In: Proceedings of the *10th International Conference on Advanced Concepts for Intelligent Vision Systems*, Berlin, Heidelberg, DE, Springer, 2008, s. 170-181, ISBN 978-3-540-88457-6.

[10] Ojala, T., Pietikäinen, M., Harwood, D.: A comparative study of texture measures with classification based on featured distributions. Pattern Recognition, ročník 29, č. 1, 1996: s. 51-59, ISSN 0031-3203.

[11] Mäenpää, T.: The Local Binary Pattern approach to texture analysis - extensions and applications. PHD thesis, Faculty of Technology, University of Oulu, 2003.

[12] Herout, A., Juránek, R., Zemčík, P.: Implementing the Local Binary Patterns with SIMD Instrictions of CPU. In proceedings of *WSCG 2010 Plzeň*, CZ, ZČU v Plzni, 2010, , ISBN 978-80-86943-86-2 p. 39-42.

[13] Zhang, L., Chu, R., Xiang, S., et al.: Face Detection Based on Multi-Block LBP Representation. In *Advances in Biometrics, Chinese Academy of Sciences*, 2007, ISBN 978-3-540-74548-8, s. 11-18.

[14] Freund. Y., Schapire, R.: A short introduction to boosting. *Soc. for Artif*, ročník 14, č. 5, 1999: s. 771-780.

[15] Holland, J.: Adaption in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, 1975.

[16] Agarwal, S., Awan, A., Roth, D.: UIUC Image Database for Car Detection.

[17] Hradiš, M., Herout, A., Zemčík, P.: Local Rank Patterns - Novel Features for Rapid Object Detection. In *Proceedings of International Conference on Computer Vision and Graphics 2008*, číslo 12 in Lecture Notes in Computer Science, Springer Verlag, 2008, ISSN 0302-9743, s. 1-12.

[18] Polok, L., Herout, A., Zemčík, P., Hradiš, M., Juránek, R., Jošth, R.: "Local Rank Differences" Image Feature Implemented on GPU. In *ACIVS '08: Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, Berlin, Heidelberg: Springer-Verlag, 2008, ISBN 978-3-540-88457-6, s. 170-181.

[19] Kadlček, F., Zemčík, P., Juránek, R.: Automatic synthesis of classifiers in FPGA. In *International Bata conference for Ph.D. Students and Young Researchers*, Tomas Bata University in Zlin, 2011, ISBN 978-80-7454-013-4.