Genetic Programming for Multiclass Texture Classification using a Small Number of Instances

Harith Al-Sahaf, Mengjie Zhang, and Mark Johnston

Evolutionary Computation Research Group,
Victoria University of Wellington, PO Box 600, Wellington, New Zealand
{harith.al-sahaf,mengjie.zhang}@ecs.vuw.ac.nz
mark.johnston@msor.vuw.ac.nz

Abstract. The task of image classification has been extensively studied due to its importance in a variety of domains such as computer vision and pattern recognition. Generally, the methods developed to perform this task require a large number of instances in order to build effective models. Moreover, the majority of those methods require human intervention to design and extract some good features. In this paper, we propose a Genetic Programming (GP) based method that evolves a program to perform the task of multiclass classification in texture images using only two instances of each class. The proposed method operates directly on raw pixel values, and does not require human intervention to perform feature extraction. The method is tested on two widely used texture data sets, and compared with two GP-based methods that also operate on raw pixel values, and six non-GP methods using three different types of domain-specific features. The results show that the proposed method significantly outperforms the other methods on both data sets.

Keywords: Genetic Programming, Texture Classification, Multiclass

1 Introduction

In the fields of computer vision and pattern recognition, *image classification* represents one of the most important tasks. However, developing a program that is capable of performing image classification with good performance is a very challenging task, particularly for difficult problems. Even discriminating between instances of two classes (binary classification) can be difficult. The difficulty of this task increases with a large number of classes due to the increased complexity of detecting a good set of features. The majority of the proposed methods for multiclass classification in the literature suffer one or both of the following issues: (1) some human intervention to design and extract a good set of features is required prior to the training phase [6]; and (2) a large number of instances are required in order to reach a suitable level of performance. In terms of the first issue, an expert with background knowledge of the domain is required to perform the task of detecting a set of distinctive features. It is not always feasible to find such a person or it can be very expensive. Similarly, in terms of the second

issue, acquiring a sufficiently large number of instances can be expensive, hard, or infeasible in many cases.

Genetic Programming (GP) is an evolutionary search method inspired by the principles of natural selection [12]. GP aims at evolving a computer program for a user-defined problem. Starting from a randomly generated initial population of solutions, GP uses genetic operators and a fitness function to evolve a solution. The fitness function is used to measure the goodness of each program, which reflects the performance level or the ability of that program to solve the problem. The genetic operators allow the system to explore or introduce different combinations of the genetic materials.

In tree-based GP [12], an evolved program is represented as a tree where all non-leaf nodes are drawn from the function set, while leaves are taken from the terminal set. The program evolved by GP produces a single value from the root node for each instance. For binary classification, the resulting value can be naturally translated to a class label such that all negative values represent one class and all non-negative values represent the other class.

There are at least four approaches that can be adopted to extend GP to perform multiclass classification tasks. In the first, a wrapper approach can be adopted where a multiclass classifier (e.g. nearest-neighbour) is used to perform the classification task, while GP is used to perform feature selection, extraction, or construction [15]. The second approach is to change the mapping scheme such that the real number line is divided into more than two intervals and the single value resulting from the root node is mapped to one of those intervals [13]. The third approach is to use a different program representation that produces multiple values instead of only a single value obtained from the root node [18]. Building a composite solution via breaking the multiclass classification task into a number of binary classification problems is the fourth approach [7].

Song et al. [13] utilised GP to perform multiclass texture classification by using *static range selection* (SRS) [16] and *dynamic range selection* (DRS) [7]. In SRS, the real number line is divided into a predefined number of equally sized intervals, each of which represents one class. This method requires $N-1$ thresholding values, where N is the number of classes. Selecting a good set of threshold values introduces an extra complexity that is an interesting research topic itself. In DRS, the idea is to dynamically divide the real number line during the program evolving phase (training). Their experiments reveal that both methods are capable of handling the multiclass classification task on texture images using the raw pixel values. SRS and DRS methods are used as baseline methods in this study.

A method that decomposes the multiclass classification task into a number of binary classification tasks was used by Loveard et al. [7]. The idea is to evolve a single program for each of the pairwise class combinations. The cost of evolving a complete set of sub-classifiers represents the main drawback of this approach.

Changing the representation of the GP program to produce multiple values from the internal nodes rather than the single value of the root node represents another approach that has been adopted to tackle the problem of multiclass

classification in [18]. The idea is to use a special type of node that make decisions to discriminate between instances of the different classes, and then a voting approach is used to predict the class label of the instance being evaluated.

In this paper, a wrapper-based approach is adopted via combining a nearest-neighbour classifier and GP to evolve a model. The main idea of the proposed method is to evolve a program that applies different operations on the instance being evaluated such that the response to a bank of filters (i.e. set of convolution masks) will be different depending on the textures of the different class instances. Therefore, a single program is evolved to handle the multiclass classification task rather than breaking the task into a number of sub-tasks.

The overall goal of this paper is to use GP to evolve a program for the task of multiclass image classification using a small number of instances. Precisely, we are interested in addressing the following questions:

- what GP representation and fitness function can be used to tackle the limited number of instances for multiclass image classification;
- whether the evolved program can compete with other GP-based methods that were also utilised to automatically handle the multiclass image classification task; and
- whether the evolved program can achieve better performance than the use of domain-specific features with commonly used classification methods.

The remainder of this paper is organised as follows. Section 2 provides a detailed discussion of the proposed method. The data sets, baseline methods, and evaluation procedure are presented in Section 3. The results of the experiments are presented and discussed in Section 4. Finally, the conclusions and recommendations for future research directions are given in Section 5.

2 The New Method

For presentation convenience, we call the proposed GP method *Tree-of-Filters* (ToFs). This section describes the terminal and function sets, fitness function, and the procedure to measure the fitness of an evolved program.

2.1 Terminal Set

The proposed method operates directly on the raw pixel values of the image, and does not require a prior step to perform feature detection and extraction. Therefore, the first component of the terminal set is the instance (image) represented as a 2D matrix. The second type in the terminal set is a list of filters that can be used to transform the image into another image via convolution. The major issue here is which filters to select as the literature shows that there are many filters of different types and sizes. We have decided to limit our scope to filters that can help in revealing the texture primitives such as lines, as presented in Figure 1. In addition to those filters, the Gaussian and Laplacian-of-Gaussian

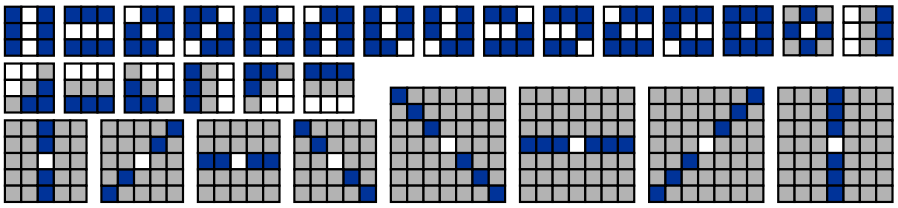


Fig. 1. The 29 filters used in this study. The blue cells are those cells having the value -1, grey cells are 0, and white cells are set to a positive value equal to the number of blue cells in order to make the sum of the filter coefficients equal to 0.

(LoG) filters were also used. The size of the Gaussian and LoG filters depends on a σ value. The size is calculated using:

$$size = (2\lceil 3\sigma \rceil) + 1 \tag{1}$$

where $\lceil \cdot \rceil$ returns the smallest integer value greater than or equal to the argument. The value of σ is randomly chosen from the set $\{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$.

The third type of terminal is a randomly generated constant value in the closed interval $[-10, +10]$.

In summary, the terminal set is made up of $\{I, F_i, G_\sigma, L_\sigma, C\}$, where I is the image being evaluated, F_i is the i^{th} predefined filter, G_σ and L_σ are respectively the Gaussian and LoG filter of the specified σ value, and C are constant values.

2.2 Function Set

The function set consists of twelve operators that can be categorised into five groups based on the number and type of the input arguments. The first group consists of the $\{Add_{\mathrm{I}}(\cdot,\cdot),\ Sub_{\mathrm{I}}(\cdot,\cdot),\ Mul_{\mathrm{I}}(\cdot,\cdot),\ Div_{\mathrm{I}}(\cdot,\cdot),\ Min_{\mathrm{I}}(\cdot,\cdot),\ Max_{\mathrm{I}}(\cdot,\cdot)\}$ functions that operates on two images and return an image. The first four functions of this group are the regular mathematical operators $\{+, -, \times, \div\}$, where the $\div$ is protected; it returns zero when the second value (divisor) is zero. The $Min_{\mathrm{I}}(\cdot,\cdot)$ and $Max_{\mathrm{I}}(\cdot,\cdot)$ functions respectively return the minimum and maximum of the arguments. The functions of this group work in a pixel-by-pixel fashion; therefore, the input and returned images are of the same size. Similarly, the second group of functions, $\{Add_{\mathrm{C}}(\cdot,\cdot),\ Sub_{\mathrm{C}}(\cdot,\cdot),\ Mul_{\mathrm{C}}(\cdot,\cdot),\ Div_{\mathrm{C}}(\cdot,\cdot)\}$, operate on an image and a constant value. The third group is the $Conv(\cdot,\cdot)$ function that convolves the first argument (image) using the second argument (filter). The fourth group is the single function $Coder(\cdot)$, which takes only one argument of type image, and returns a feature vector. This is only used at the root of the evolved program. Each function of the first three groups normalises the resulting image to have pixel values between 0 and 255 (inclusive) before passing it to the parent node. Moreover, those functions, apart from $Coder(\cdot)$, can form long chains in different orders due to type matching between their outputs and at least one of the input arguments. Figure 3 shows two examples of programs evolved by the ToFs method.

Algorithm 1: Distance ratios (DR_{diff} and DR_{same})

Input: Ω : list of lists of Imprints
Output: Set of double-precision values

$\Sigma_{\text{differ}} \longleftarrow 0$
$\Sigma_{\text{same}} \longleftarrow 0$
$\xi \longleftarrow 0$ // ξ is a counter
foreach $\omega_1 \in \Omega$ **do** // ω_1 is a list of Imprint objects
  foreach $\nu_1 \in \omega_1$ **do** // ν_1 is an Imprint object
    $D_{\text{differ}} \longleftarrow 2$
    $D_{\text{same}} \longleftarrow -1$
    foreach $\omega_2 \in \Omega$ **do** // ω_2 is a list of Imprint objects
      foreach $\nu_2 \in \omega_2$ **do** // ν_2 is an Imprint object
        $\lambda \longleftarrow \text{dist}(\nu_1,\nu_2)$
        if $(\omega_1 \neq \omega_2) \wedge (\lambda < D_{\text{differ}})$ **then** $D_{\text{differ}} \longleftarrow \lambda$
        else if $(\nu_1 \neq \nu_2) \wedge (\lambda > D_{\text{same}})$ **then** $D_{\text{same}} \longleftarrow \lambda$
      end
    end
    $\Sigma_{\text{differ}} \longleftarrow \Sigma_{\text{differ}} + D_{\text{differ}}$
    $\Sigma_{\text{same}} \longleftarrow \Sigma_{\text{same}} + D_{\text{same}}$
  end
  $\xi \longleftarrow \xi + |\omega_1|$
end
return $\{\Sigma_{\text{differ}}/\xi, \Sigma_{\text{same}}/\xi\}$

2.3 Fitness Measure

The fitness function of the ToFs performs multiple tasks simultaneously as shown in Equation (2).

$$Fitness = DR_{\text{same}} + \big(2 - (DR_{\text{diff}} + Accuracy)\big) \tag{2}$$

$$Accuracy = \frac{Hits}{Total} \tag{3}$$

Here *Accuracy* measures the performance (accuracy) of the individual on the training set, and *Hits* and *Total* are respectively the number of correctly classified instances and total number of instances. The DR_{same} and DR_{diff} are the distance ratio to instances of the same and different class respectively. Algorithm 1 presents the procedure for calculating the DR_{same} and DR_{diff} values.

The $\text{dist}(\cdot,\cdot)$ function in Algorithm 1 calculates the distance between two feature vectors of the same length:

$$\text{dist}\big(a,b\big) = 1 - \left(\frac{2\left(\sum_{i=1}^{|a|} min\left(a_i,b_i\right)\right)}{\sum_{i=1}^{|a|} a_i + \sum_{i=1}^{|b|} b_i}\right) \tag{4}$$

where a and b are the two feature vectors, the $|\cdot|$ function returns the length (i.e. number of elements or items) of a vector, $min(\cdot,\cdot)$ returns the minimum value of the two arguments, a_i and b_i are the value of the i^{th} feature of a and b respectively. This distance measure returns a value between 0 and 1, where a smaller value means a higher similarity between the two feature vectors.

2.4 Fitness Measuring Procedure

The training phase aims at evolving a program that has high accuracy on the training set, high distance ratio between the instances of different classes, and

low distance ratio between instances of the same class. The evaluation of the program starts from the terminal nodes as they represent the inputs of the program's tree, and ends at the root node (i.e. *Coder*). For each instance in the training set, the system applies the operations in a bottom-up order. At each non-terminal node, an image is generated depending on the inputs and the specified operation. Then this image is normalised (to have values between 0 and 255) and passed to the parent node. The *Coder* node then receives the final image and transforms it to a feature vector. The *Coder* node has a filter-bank (list of filters) identical to those of the terminal set apart from the Gaussian and LoG filters. This node constructs a new feature vector (all elements have zero value) which consists of a number of elements equivalent to the number of filters in the filter-bank (one element for each filter). A dot product is then performed at each pixel of the image argument, along with the neighbouring pixels, with each of the filters in the filter-bank, and the responses (resulting values) are reported. The corresponding element of the filter that has the highest response (largest value) is incremented by one. Finally, the generated feature vector is normalised to have values between 0 and 1.

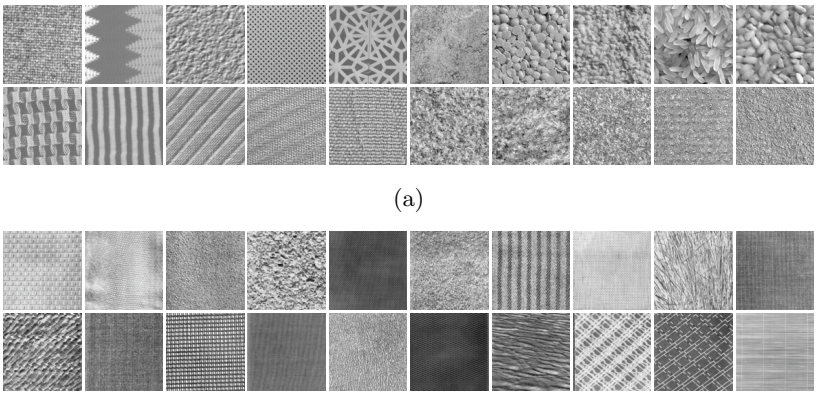
The system uses the feature vector generated by the *Coder* node along with the actual class label of the instance being evaluated to construct an *Imprint* object. The constructed imprint objects of the training set instances are stored in a list named *knowledge base* that will be used for two tasks: (1) to measure the fitness of the evolved program; and (2) to serve as a knowledge base during the testing phase. The DR_{same} and DR_{diff} values are first calculated using the procedure presented in Algorithm 1. Meanwhile, the accuracy is measured using *the-nearest-neighbour* (1NN) [2] method. Using those three values (DR_{same}, DR_{diff}, and accuracy), the fitness of the evolved program can be measured using the formula presented in Equation (2).

2.5 Performance Measuring Procedure

The aim of the testing phase is to measure the generalisation ability of the best evolved program on the unseen data. Therefore, the best evolved program at the end of the training phase is tested on the instances of the test set (unseen data). For each instance, a feature vector is generated from the *Coder* node in a similar way to that in the training phase. The distance between the generated feature vector and each imprint object of the *knowledge base* list is calculated. The class label of the closest imprint object is returned to serve as the predicted class label for the instance being evaluated. Then the accuracy formula (Equation (3)) is used to measure the generalisation ability of the best evolved program.

3 Experimental Design

In this section, discussions of the data sets, data set preparation, baseline methods, parameter settings, and the evaluation process are provided.



(a)

(b)

Fig. 2. Samples of the (a) Textures-1 data set; and (b) Textures-2 data set.

3.1 Data Sets

In this study, two data sets are used to evaluate the performance of the ToFs method. The first data set is taken from the *Kylberg Texture* data set [5], which is made up of 28 grey-scale texture classes. Each class consists of 160 instances of size 576×576 pixels. The instances of this data set are fixed in terms of rotation and scale, but not illumination. This data set comes in another flavour where the instances are captured under different rotation angles (not used in this study). Only 20 classes of the 28 have been selected to form the first data set of this study *Textures-1* as presented in Figure 2(a). The instances of this data set have been resampled (i.e. resized) to 115×115 pixels to reduce the computation costs.

In computer vision and signal processing, the *Brodatz textures* data set [1] is one of the mostly used data sets. This data set is made up of 112 classes of different textures that each consists of only one grey-scale instance of size 640×640 pixels. Similar to *Textures-1*, 20 classes have been selected to form the second data set in this study *Textures-2* as presented in Figure 2(b). The original image of each class has been divided into 16 distinct sub-samples each of size 160×160 pixels.

3.2 Baseline Methods

In this study, two GP-based and four non-GP methods have been used as the baseline methods for comparison purposes.

GP Methods

– Static Range Selection (SRS) [16]: in the SRS method, the real number line is divided into a number of equally and fixed size intervals, where each interval is allocated for one classes. The SRS method uses the accuracy function to measure the fitness of an evolved program.

– Dynamic Range Selection (DRS) [7]: the DRS method is similar to SRS in terms of program representation, terminal and function sets, and fitness measure. However, the real number line is divided into intervals dynamically rather than using predefined intervals. Moreover, in the DRS method, the training set is divided into *segmentation* and *evaluation* sets. The former is used to define the corresponding interval of each class, while the latter is used to measure the fitness of the evolved program.

Non-GP Methods

– Naive Bayes (NB) [14]: NB is a simple, yet powerful, classifier that uses Bayes' theorem to build a decision model.
– Support Vector Machines (SVM) [14]: SVM is a broadly used classifier in the literature. A SVM is trained using algorithm of Platt [11] named *sequential minimal optimisation* (SOM).
– Naive Bayes / Decision Trees (NBTree) [14]: Combines Decision Trees (DT) with NB method to form a hybridised method that inherits the characteristics of the two methods. DT is used to build the tree, whilst NB is used at the leaves of the tree.
– K* (KStar) [14]: similar to 1NN, KStar predicts the class label of an instance based on the similarity to the closest instance in the training set. The KStar method uses an entropy-based distance measure to calculate the distance between two instances.
– Non-nested generalized (NNge) [14]: similar to 1NN, NNge is an instance-based classifier that operates based on similarity measure. Moreover, NNge uses a non-nested exemplar.
– Multilayer Perceptron (MLP) [14]: A artificial multilayer neural network trained using the back-propagation algorithm.

3.3 Data Sets Preparation

In both data sets, the total number of instances of each class has been divided equally between the training and test sets. Moreover, the instances of the two data sets are *standardised* to have zero mean and unit standard deviation. The standardised images then are *normalised* to have values between 0 and 255. The three GP-based methods do not require feature detection and extraction as they were designed to operate directly on raw pixel values. However, all non-GP methods require performing feature detection and extraction in a prior stage. Three different feature extraction methods have been used in this study: (1) domain-independent features (DIF) [17]; (2) Haralick texture features [4]; and (3) local binary patterns (LBP) [10]. In the first method, each instance has been divided into five regions that are the four quadrants and the center area of the image. The mean and standard deviation values of each of these five regions and the entire image are calculated to form the feature vector. Therefore, the feature vector of each image consists of twelve values. The second method is based on the use of the *grey-level co-occurrence matrix* (GLCM), which represents a

Table 1. The GP Parameters of all experiments

Parameter	Value	Parameter	Value	Parameter	Value
Crossover Rate	0.80	Generations	30	Selection Type	Tournament
Mutation Rate	0.19	Population Size	100	Tournament size	7
Elitism Rate	0.01	Tree depth	2-10	Initial Population	Ramped half-and-half

very popular method to extract texture features. In this study, the matrices are generated using the four orientations $\{0°, 45°, 90°, 135°\}$, of one pixel distance, and a full range (8-bits) of grey-levels. Hence, each matrix is of size 256×256. The third method, LBP, is a dense-based feature descriptor that has been used extensively in the literature to extract texture features. In our experiments, each instance has been transformed into a histogram of uniform $LBP_{8,1}$ codes [10].

3.4 Parameter Settings and Implementation

To draw fair conclusions, all experiments have been conducted under the same conditions. The parameter settings of GP-based methods are shown in Table 1.

The three GP-based methods (one proposed and two baseline) have nodes that vary in the types of inputs and outputs, and the number of input arguments. Therefore, *Strongly-typed Genetic Programming* (STGP) [9] is required to implement these methods. The *Evolutionary Computation Java-based* (ECJ) package [8] is used to implement STGP based methods. The *Waikato Environment for Knowledge Analysis* (WEKA) package [3] has been used to evaluate the non-GP methods on the two data sets.

3.5 Evaluation

Only two instances of each class are randomly selected to form the training set. Similar to other stochastic search methods, GP produces different results based on the seed to the random number generator. Hence, the process of evolving a program has been repeated 30 times independently and using different random seeds. The average performance of the best evolved programs on the test set at the end of the 30 runs is then reported. The non-GP methods, apart from MLP, that were used in this study are all deterministic. Therefore, each of them has been tested only one time; while the average performance for 30 runs of MLP is reported. The selected instances forming the training set have a great impact on the final result. Hence, and as only two instances are used, the same procedure of 30 independent runs has been further repeated 10 times using different instances in the training set each time. The average performance of those 10 repetitions along with the standard deviation is reported.

4 Results and Discussions

The results of the experiment are presented and discussed in this section. The two-tailed unpaired t-test is used to check whether the difference between the performance of the proposed method compared to that of the baseline methods is significant or not. The significance level of the t-test is set to 5%. The superscript "$*$" appears if ToFs has significantly outperformed the other method.

Table 2. Accuracies of the Textures-1 data set.

	SRS		DRS		ToFs	
	5.02 ± 0.21*		9.84 ± 8.01*		94.31 ± 1.39	
Features	NB	SVM	NBTree	KStar	NNge	MLP
DIF	26.07 ± 3.51*	36.27 ± 4.20*	31.06 ± 5.60*	27.73 ± 2.39*	40.63 ± 5.12*	34.71 ± 3.72*
Haralick	70.77 ± 9.12*	84.43 ± 3.85*	71.81 ± 4.10*	84.11 ± 4.21*	86.48 ± 2.86*	81.86 ± 3.26*
LBP	75.80 ± 6.57*	82.17 ± 3.58*	78.52 ± 7.17*	86.82 ± 2.92*	88.68 ± 2.14*	85.54 ± 3.09*

Table 3. Accuracies of the Textures-2 data set.

	SRS		DRS		ToFs	
	5.80 ± 1.34*		2.28 ± 0.21*		95.74 ± 1.90	
Features	NB	SVM	NBTree	KStar	NNge	MLP
DIF	46.64 ± 5.80*	58.21 ± 6.36*	53.67 ± 10.22*	61.56 ± 5.49*	59.77 ± 4.67*	56.49 ± 7.42*
Haralick	84.22 ± 4.31*	90.78 ± 1.73*	80.00 ± 5.19*	89.61 ± 4.01*	92.97 ± 2.79*	92.58 ± 3.97*
LBP	83.68 ± 3.76*	89.53 ± 3.16*	84.46 ± 6.89*	90.86 ± 3.24*	90.71 ± 3.49*	92.19 ± 3.36*

Each of the tables presented in this section is divided vertically into two blocks. The upper block presents the results of the GP-based methods, while the results of the non-GP methods are presented in the lower block. Moreover, three values are listed under each of the non-GP methods that each corresponds to one of the three features extraction methods were discussed in Section 3.3.

4.1 Overall Results

The results on the Textures-1 data set are presented in Table 2. The statistical test shows that ToFs has significantly outperformed all other methods on this data set. Both of the GP-based baseline methods show very poor performance on this data set. The use of hand crafted features with the six non-GP methods shows a good level of performance. A lower level of performance has been achieved when the DIF features are used by all those methods compared to LBP and Haralick. In most of the cases, the use of LBP features results in a slightly better performance than that of the Haralick features.

Table 3 presents the results on the Textures-2 data set. Similar to Textures-1, ToFs has significantly outperformed all other methods on this data set. SRS and DRS have achieved the lowest accuracies amongst all other methods. Similar to Textures-1, the non-GP methods show poor performances when the DIF features are used. Moreover, those methods achieved a good level of performance when LBP features or Haralick features are used.

4.2 Analysis

The results show that the simple domain-independent features are not sufficiently powerful for these data sets. Moreover, GP with SRS and DRS are not suitable for classification when the number of classes is large. These two methods simply translate the single floating number output into a set of class labels, while the proposed method evolves a program that implicitly performs feature extraction and generates a powerful feature vector.

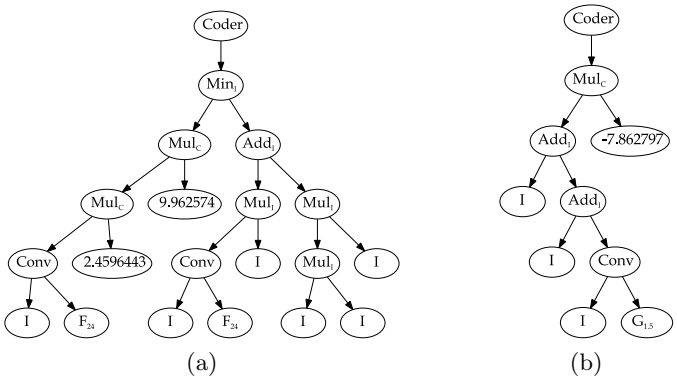


Fig. 3. Sample programs evolved on (a) Textures-1, and (b) Textures-2 data sets.

Figure 3(a) shows a program that was trained on the Textures-1 data set. This program has scored 95.50% accuracy on the unseen data. Meanwhile, a program evolved by the proposed method on the Textures-2 data set is presented in Figure 3(b). This program has scored 100% accuracy on the unseen data. It convolves the image with a Gaussian filter with $\sigma = 1.5$, then adds it to the original image twice. The resulting image is then multiplied by a constant value (-7.862797) and passed over to the root node to generate the feature vector.

5 Conclusions

In this paper, a GP-based method has been proposed for the task of multiclass classification in texture images. The proposed method uses only two instances of each class to evolve a program that operates on raw pixel values. Two well-known data sets have been used to evaluate the performance of the proposed method. Moreover, the performance achieved has been compared to that of two GP-based and six non-GP methods. Similar to the proposed method, the two GP baseline methods operate on raw pixel values to perform multiclass texture classification. The non-GP methods, on the other hand, require a set of pre-extracted features to build a model. Therefore, three different feature extraction methods have been used and the performances obtained have been compared to that of the proposed method. The results of the experiments show that the proposed method significantly outperformed all other methods on both of the data sets used.

In the future, we would like to test the ability of the method to handle rotation and scale variants, and on different domains other than textures. Analysing some of the evolved programs to highlight some important patterns and to investigate the costs (i.e. speed and memory) is another objective to investigate in the near future.

References

1. P. Brodatz. *Textures: A Photographic Album for Artists and Designers.* Dover Publications, 1999.

2. E. Fix and J. Hodges. Discriminatory analysis-nonparametric discrimination: Consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolf Field, Texas, 1951.
3. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
4. R. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-3(6):610–621, 1973.
5. G. Kylberg. The Kylberg texture dataset v. 1.0. External report (Blue series) 35, Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, Uppsala, Sweden, 2011.
6. I. Levner, V. Bulitko, L. Li, G. Lee, and R. Greiner. Automated feature extraction for object recognition. In *Proceedings of the Image and Vision Computing New Zealand*, pages 309–314. Massey University, 2003.
7. T. Loveard and V. Ciesielski. Representing classification problems in genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 2, pages 1070–1077. IEEE Press, 2001.
8. S. Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013.
9. D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
10. T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.
11. J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, pages 185–208. MIT Press, 1999.
12. R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu, 2008. (With contributions by J. R. Koza).
13. A. Song, T. Loveard, and V. Ciesielski. Towards genetic programming for texture classification. In M. Stumptner, D. Corbett, and M. Brooks, editors, *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence*, volume 2256 of *Lecture Notes in Computer Science*, pages 461–472. Springer, 2001.
14. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann, San Francisco, CA, USA, 2005.
15. L. Zhang, L. Jack, and A. Nandi. Extending genetic programming for multiclass classification by combining k-nearest neighbor. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages v/349–v/352, 2005.
16. M. Zhang and V. Ciesielski. Genetic programming for multiple class object detection. In N. Foo, editor, *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, volume 1747 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 1999.
17. M. Zhang, V. B. Ciesielski, and P. Andreae. A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Advances in Signal Processing*, 2003(8):841–859, 2003.
18. M. Zhang and M. Johnston. A variant program structure in tree-based genetic programming for multiclass object classification. In S. Cagnoni, editor, *Evolutionary Image Analysis and Signal Processing*, volume 213 of *Studies in Computational Intelligence*, pages 55–72. Springer, 2009.