Texture Analysis by Genetic Programming

Andy Song
School of Computer Science and Information Technology
RMIT University
Melbourne 3001, Australia
Email: asong@cs.rmit.edu.au

Vic Ciesielski
School of Computer Science and Information Technology
RMIT University
Melbourne 3001, Australia
Email: vc@cs.rmit.edu.au

Abstract—This paper presents the use of genetic programming (GP) to a complex domain, texture analysis. Two major tasks of texture analysis, texture classification and texture segmentation, are studied. Bitmap textures are used in this investigation. In classification tasks, the results show that GP is able to evolve accurate classifiers based on texture features. Moreover by using the presented method, GP is able to evolve accurate classifiers without extracting texture features. In texture segmentation tasks, the investigation shows that a fast and accurate segmentation method can be developed based on GP generated texture classifiers. Our further investigation show that the accuracies are not achieved by chance. There are regularities been captured by GP-generated classifiers in performing texture discrimination.

I. INTRODUCTION

Genetic programming (GP) has emerged as a flexible problem solving method for a wide range of complex problems [5]. It has shown to be effective in a variety of domains such as medical diagnosis [6], object detection, and image analysis [9], [11]. In our investigation, genetic programming has been used as a new method to address a complex domain: texture analysis.

One task that GPs have previously been applied to is that of classification.

In classification tasks, a classifier is evolved from a given set of examples where the class of the example is known. The accuracy of the classifier is determined by the application of the resulting classifier on unseen data, known as test data. In previous investigations GPs have shown to be capable of producing accurate classifiers in a variety of domains such as medical diagnosis [6], [7], object detection and image analysis [9], [11].

Texture is observed as homogeneous visual patterns of scenes that we perceive, such as grass, cloud, wood and sand. The repetition of such patterns somehow produces the uniformity of sense which is very important for an observer to understand the scenes, for example partitioning two different kinds of background or recognising objects from a textured background. The textural property of an image is one of the most informative cues for machine vision tasks. However, there is no universally accepted definition of texture and no universal model to describe texture. Analysing texture information of images still remains a complex problem.

There are two main texture analysis problems, texture classification and texture segmentation. In texture classification, image textures are categorized into different classes and an observed image will be determined to belong to one of the given set of texture classes. The conventional method of texture classification usually involves two steps. The first step is for obtaining a priori knowledge of each class to be recognized. This process is often known as texture feature extraction. Once the texture features of the observed image are extracted, then classical classification techniques, e.g. nearest neighbors and decision trees, can be used to make the decision [10]. The fields that texture classification has been applied to include the classification of satellite images [8], radar imagery [1], inspection [3] and content based image retrieval [4].

In this paper we use genetic programming as a classification method in texture classification. Furthermore, GP is used to evolve texture classifiers directly based on raw pixel data. In the second approach, the process of feature extraction is not required. As a result it does not require manual interruption or detailed domain knowledge. To differentiate these two methods, we refer the first method as two-step approach and the second one as single-step approach.

Texture segmentation, which can be considered as an extension of texture classification, is the overlapping area between texture analysis and image segmentation. The aim of texture segmentation is to partition an image into regions based on differences of textural appearance. In some sense, texture segmentation has more practical values compared to texture classification, especially in machine vision applications. To show the feasibility of GP in this area, we use texture classifiers generated by single-step approach to perform segmentation tasks.

In this paper, our investigation has been described in three parts. The first part addresses texture classification, which contains the discussions of two-step approach and single-step approach. The second part addresses texture segmentation. The third part is the analysis of the generated-classifiers, which is to understand the behaviors of evolved classifiers and to show whether there are regularities been captured. The aim of our research investigation is to explore the applications of GP towards the texture analysis problems, and to determine GP methodology that can be appropriately applied to this complex domain. Additionally this work aims to provide support for the suitability of the GP approach to the texture domain and for the applicability of the method in general.
II. TEXTURE DATA SET

There are almost infinite numbers of texture available. In this investigation, bitmap patterns are used. Our investigation on more complex textures, such as grey level textures, is not included in this paper. Bitmap textures are composed of pixels of only two values, 0 or 1. The dataset consists of 48 different bitmap textures. Snapshots of each textures are presented in Figure 1, named from P11 to P68. They are widely used in Microsoft Windows applications. Unlike arbitrary textures, these textures can be easily generated without variation. The easy accessibility of these patterns allows researchers to conveniently re-create these images and perform experiments.

![Fig. 1. Bitmap Patterns](image)

The simplicity of these bitmap patterns does not mean that they are not good research objects. In fact classifying these patterns is not as easy as it seems. For example P14 is formed by vertical lines and P24 is formed by horizontal lines, but the thickness of the lines and the distance between two adjacent lines in the two patterns are identical. So a texture classification method based on first-order statistics, or not sensitive to orientation, will not be able to differentiate them. Bitmap textures also have significant application value. In some image analysis tasks, the input data for a classification process are binary images. For example an input gray-level image could be pre-processed by thresholding, which not only filters out noise but also reduces computation requirements. The resulting image is a binary image. If a pixel intensity value is higher than the threshold then it will be treated as 1. Otherwise the pixel will be treated as 0. Furthermore the images generated by some sensing devices are binary in which 0 represents the absence of a signal in that position and 1 represents the presence. The capability of handling these bitmap textures will be desirable for such applications.

III. TEXTURE CLASSIFICATION

A. Feature Extraction

Haralick features, also known as Gray Level Co-occurrence Matrix (GLCM) features, are one of the most widely used texture features. In our investigation thirteen Haralick feature functions were computed. They are (1) Angular Second Moment, (2) Contrast, (3) Correlation, (4) Variance, (5) Inverse Difference Moment, (6) Sum Average, (7) Sum Variance, (8) Sum Entropy, (9) Entropy, (10) Difference Variance, (11) Difference Entropy, (12) and (13) Mean of Correlation. Every function is applied at four angles for a certain distance value, which are 0°, 45°, 90° and 135° respectively. At each distance, the average of each feature computed at the four angles is used as an additional feature. The distance values are set as 1, 3 and 5. Therefore the total number of Haralick features for bitmap patterns are: 13 Functions × 5 Angles × 3 Distances = 195 Features.

B. Two-step Approach

1) The Fitness Function: The fitness measure for GP classification tasks is straightforward. It is determined by the classification accuracy which is expressed as the following formula:

\[
\text{fitness} = \frac{\text{Correct Classification}}{\text{TOTAL}} \times 100\% \quad (1)
\]

where TOTAL is the total number of instances in training. Correct Classification is the number of these sub-images or images that were correctly classified. The training accuracy of a generation refers to the accuracy achieved by the individual which has the highest fitness in that generation of training. The fitness calculation given by equation 1 is also used to measure the performance of generated classifiers on test data.

2) The Function Set: The main considerations in selecting functions are:

- the candidate functions should be able to perform the basic arithmetic and logical operations available in programming languages.
- the candidate functions should be computationally inexpensive, otherwise executing the generated programs could require enormous resources.
- The set of functions should be rich enough to permit a solution to be evolved.

Followed the above criteria, there are two categories of functions in the function set: arithmetical and logical. The selected ones are +, −, *, , IF, >=, <=, =, Between.

3) The Terminal Set: Terminals are the program inputs. These are shown in Table I. The “Return Type” column indicates the data type of the values returned by these terminals. There are two types of terminals in the table. The first type is a “Random” terminal, which is used to generate random...
numbers in the range of -1 to 1. These random numbers can behave like parameters in mathematical functions to adjust the weight of a part of the function, or to set a bias, for example, $0.231 \times (x+y-0.485)$. The second is the “Feature[x]” terminal which reads in and returns the value of the $x$th feature in a feature vector.

<table>
<thead>
<tr>
<th>Name</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random(-1,1)</td>
<td>Double</td>
<td>Random constant [-1,1]</td>
</tr>
<tr>
<td>Feature[x]</td>
<td>Double</td>
<td>Value of Feature x</td>
</tr>
</tbody>
</table>

**TABLE I**

**TERMINAL SET OF TWO-STEP APPROACH**

4) Main Parameters and Termination Conditions: Table II presents the major runtime parameters chosen for the experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bitmap Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossover Rate</td>
<td>85%</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>5%</td>
</tr>
<tr>
<td>Elitism Rate</td>
<td>10%</td>
</tr>
<tr>
<td>CROSS$_$CHANCE$_$TERM</td>
<td>10%</td>
</tr>
<tr>
<td>CROSS$_$CHANCE$_$FUNC</td>
<td>90%</td>
</tr>
<tr>
<td>INITIAL$_$MAX$_$DEPTH</td>
<td>6</td>
</tr>
<tr>
<td>MAX$_$DEPTH</td>
<td>17</td>
</tr>
<tr>
<td>Population Size</td>
<td>200</td>
</tr>
<tr>
<td>NUM$_$GENERATIONS</td>
<td>50</td>
</tr>
</tbody>
</table>

**TABLE II**

**RUNTIME PARAMETERS**

The evolutionary process for producing texture classifiers will be terminated when one of the following conditions is met:

- The classification problem has been completely solved, that is, a perfect classifier has been found which can correctly differentiate all texture images and has 100% accuracy.
- The maximum number of generations, NUM$\_\$GENERATIONS, is reached.

C. Single-step Approach

Table III lists the terminal set of single-step approach. Instead of feature terminals shown in Table I, the second type of terminals of this approach read pixel value directly. The process of extracting texture features is not required here. Other than terminal set, the other aspects of single-step approach are identical to that of two-step approach, such as function set, fitness function and run time parameters. This is to show the difference between these two approaches, in terms of classification performance.

The main reasons for considering the single step approach are:

- Unlike other classification methods, GP constructs classifiers by generating programs. This suggests that the generated classifiers could use pixel data to directly classify textures, since feature extraction and classification are also implemented as programs.
- In texture analysis, numerous feature extraction schemes have been proposed. However none of the schemes is universally applicable due to the lack of knowledge of the true nature of texture. In this situation, genetic programming might offer a new direction since it can evolve programs towards a solution without pre-defined domain knowledge.
- Designing a texture classification system is time consuming. It usually involves selecting a good combination of “feature extraction + classification” and optimizing the chosen feature extractor in addition to the chosen classifier. It will be economically favorable if a program to achieve texture classification without user intervention can be evolved automatically.
- Current texture feature extraction methods usually require considerable resources to compute the features before the classifier can be applied. Even if a classifier is fast, the overall process is still computationally expensive. This limits the application of such methods in some circumstances, such as real-time applications and applications with a small resource environment. In contrast, the programs evolved by genetic programming are relatively simple, small and execute quickly. Only the creation of such a texture classifier will be resource intensive.

D. Experiments and Results

In our investigation, the experiments only focus on one kind of binary classification which is to discriminate one texture from a group of other textures. In one experiment the sub-images from P11 are considered as one class while the sub-images from P12, P13 ... P68 are treated as the other class. The goal of this experiment is differentiate P11 from other bitmap textures. This experiment is done for each textures, so there is a total of 48 such experiments for 48 textures. The experimental data for each pattern comprised 1034 sub-images sampled from the chosen pattern and 22 sub-images from each of the 47 patterns in the dataset. Using P11 as an example again, there were 1034 cutouts generated from P11 and 1034 (22 x 47) sub-images generated from P12 to P68. The sampling window size used for bitmap patterns was 8 x 8.

For comparison purposes, conventional classification methods were used to classify these textures as well. Among six well-known methods, C4.5 generally achieved the highest accuracy. Therefore the results obtained by C4.5 are used in this paper. Other methods are PART, One Rule, Naive.
Bayesian classifier, Decision Tables and Instance Based classifier. Table IV lists the classification accuracies achieved by three approaches. The rows labeled with "FE + C4.5" are the accuracies obtained by C4.5 based on texture features. The rows labeled with "FE + GP" are the accuracies obtained by the two-step approach, which is classifying by GP based on texture features. The third result in each cell is obtained by the single-step approach. These results shown in Table IV are all average test accuracies from ten-fold cross validation.

The experiment results show that C4.5 achieved perfect classification on 40 out of 48 cases. In the remaining 8 cases, its accuracy was close to 100%. However genetic programming produced even better results in the two-step approach ("FE + GP"). Although the feature vectors have 195 attributes, perfect classification of 100% average test accuracy was achieved in all 48 cases. The GP training processes for the 48 cases were all able to find the perfect solution and terminated early. It was observed that the ten runs for each case were terminated in the first generation. This observation means that in all 480 (48 × 10) runs, the GP method was able to generate at least one classifier, with 100% accuracy on training data.

In case of single-step approach, the accuracies were slightly lower than that of other two methods. There were only five cases that the accuracy was lower 90%. There were five cases that the accuracy was in the range of 90% ~ 95%. In the rest 38 cases out of the 48 cases, a test accuracy above 95% was achieved. In more than one third of cases, 18 cases, the accuracy was above 99%.

Table IV also shows the average accuracies of the three approaches, which are 99.97%, 100% and 96.84% respectively. Students T-tests were performed which indicated the differences were statistically significant. Such results suggest that GP can be applicable on texture classification tasks and can evolve accurate classifiers based on texture features. In general, the single-step approach was less accurate. However its results suggest that evolving texture classifiers directly based on raw pixels is possible. Relatively low accuracy does not mean this approach has no merit. In the next section, it is shown that single-step approach can be used to address texture segmentation.

IV. TEXTURE SEGMENTATION

The segmentation algorithm developed based on the single-step approach is shown in Figure 2. The first step is evolving texture classifiers by the single-step approach described in previous section. The reason of adapting the single-step approach is that the time-consuming feature extraction phrase can be removed and no need to consider the optimization between feature extraction and classification. The segmentation algorithm is a supervised method because sub-images need to be labeled for training. Although GP can be used for unsupervised learning such as [2], it is more practical to use it as a means of supervised learning in our study. This algorithm can be considered as a region-based approach, because it is based on labeling regions rather than differentiating two adjacent

<table>
<thead>
<tr>
<th>Input:</th>
<th>T1, T2 textures for which example images are available</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>an image containing a number of regions of textures T1 and T2 and no other textures</td>
</tr>
<tr>
<td>w,h</td>
<td>width and height of I</td>
</tr>
<tr>
<td>n</td>
<td>sub-image size</td>
</tr>
<tr>
<td>d</td>
<td>step size for moving window, 1 &lt; d ≤ n/2</td>
</tr>
<tr>
<td>Output:</td>
<td>O a two colour image O_{1j} = colour1 for texture 1 and O_{2j} = colour2 for texture 2</td>
</tr>
</tbody>
</table>

1) Generate a single-step classifier which uses a sub-image of size n × n.
2) Use the generated classifier to sweep the input image I.
   a) Start at the top-left corner of the image.
   b) Sample a sub-image by a n×n window and classify it by the generated classifier.
   c) Label all the pixels in this window with the class label output of the classifier.
   d) Move the window d pixels right and repeat 2(b) and 2(c), until the window reaches the right boundary of the image.
   e) Reposition the window to the left edge of the image and move down by d pixels, then repeat from step 2(b) until the whole image has been completely sampled.
3) Generate the output image which only contains the regions.
   a) Label each pixel based on voting. For example, if the majority of classifier outputs for a pixel is texture T1, then this pixel is labeled as T1.
   b) Assign each pixel a color based on its class label.
   c) Output the generated image O.

Fig. 2. Segmentation Algorithm for Two Textures

regions. A voting strategy is used to determine the class of a pixel.

In the rest of this section, some examples of texture segmentation tasks are presented, including segmenting multiple regions and segmenting regions with complex boundaries. All the segmentation experiments were conducted on a SUN Sparc workstation. The CPU runtimes on this machine are presented with the corresponding output images.

A. Two textures with Multiple Regions

Figure 3 shows an input image containing two bitmap textures (vertical lines and horizontal lines) and the segmented output image. The parameter values are also shown. The texture regions in the output image are identified by two different intensity levels, gray and black. Based on visual comparison between the original image and the corresponding output, the segmentation performance is very good. Our study aims to
investigate the methodology rather than to compare it with other methods in terms of segmentation accuracy. Therefore a quantitative measurement of segmentation performance is not used.

The CPU time measured was 0.07 seconds, which indicates that the classifier generated by the single-step approach is very fast on segmenting a 256 × 256 binary image. The boundary between the left half and the right half in the output image is precisely the boundary in the original image. The four small patches of “alien” textures are also identified and accurately located although their boundaries are not perfectly square.

The input image in Figure 4 has a butterfly-shaped boundary. The left wing of the butterfly contains the same texture as the background of the right half of the image, while its right wing has the same texture as the left sided background. The same classifier used for segmenting image in Figure 3 was also used here. The CPU time was also 0.07 seconds. As expected, the CPU runtime is independent of the content of target images.

V. ANALYZING TEXTURE CLASSIFIERS

The investigation in the previous sections shows that GP can be used to evolve classifiers for performing texture classification and segmentation tasks. However we have little knowledge about these evolved classifiers themselves, such as how did they achieve accurate classification and what kinds

<table>
<thead>
<tr>
<th>Pattern</th>
<th>FE + C4.5</th>
<th>FE + GP</th>
<th>Single-step</th>
</tr>
</thead>
<tbody>
<tr>
<td>P11</td>
<td>100%</td>
<td>99.73%</td>
<td>99.35%</td>
</tr>
<tr>
<td>P12</td>
<td>100%</td>
<td>99.8%</td>
<td>99.34%</td>
</tr>
<tr>
<td>P13</td>
<td>100%</td>
<td>99.8%</td>
<td>99.67%</td>
</tr>
<tr>
<td>P14</td>
<td>100%</td>
<td>99.89%</td>
<td>98.47%</td>
</tr>
<tr>
<td>P15</td>
<td>100%</td>
<td>100%</td>
<td>96.91%</td>
</tr>
<tr>
<td>P16</td>
<td>100%</td>
<td>100%</td>
<td>90.6%</td>
</tr>
<tr>
<td>P17</td>
<td>100%</td>
<td>100%</td>
<td>97.45%</td>
</tr>
<tr>
<td>P18</td>
<td>100%</td>
<td>100%</td>
<td>99.78%</td>
</tr>
<tr>
<td>P21</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P22</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P23</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P24</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P25</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P26</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P27</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P28</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P31</td>
<td>97.35%</td>
<td>99.73%</td>
<td>99.33%</td>
</tr>
<tr>
<td>P32</td>
<td>99.89%</td>
<td>100%</td>
<td>99.72%</td>
</tr>
<tr>
<td>P33</td>
<td>99.89%</td>
<td>100%</td>
<td>97.96%</td>
</tr>
<tr>
<td>P34</td>
<td>100%</td>
<td>100%</td>
<td>98.55%</td>
</tr>
<tr>
<td>P35</td>
<td>98.74%</td>
<td>100%</td>
<td>87.25%</td>
</tr>
<tr>
<td>P36</td>
<td>99.39%</td>
<td>100%</td>
<td>95.96%</td>
</tr>
<tr>
<td>P37</td>
<td>99.72%</td>
<td>100%</td>
<td>99.43%</td>
</tr>
<tr>
<td>P38</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P41</td>
<td>99.78%</td>
<td>99.78%</td>
<td>98.78%</td>
</tr>
<tr>
<td>P42</td>
<td>100%</td>
<td>100%</td>
<td>96.77%</td>
</tr>
<tr>
<td>P43</td>
<td>100%</td>
<td>100%</td>
<td>97.32%</td>
</tr>
<tr>
<td>P44</td>
<td>100%</td>
<td>100%</td>
<td>82.32%</td>
</tr>
<tr>
<td>P45</td>
<td>100%</td>
<td>100%</td>
<td>98.49%</td>
</tr>
<tr>
<td>P46</td>
<td>99.84%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P47</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P48</td>
<td>98.00%</td>
<td>100%</td>
<td>99.44%</td>
</tr>
<tr>
<td>P51</td>
<td>99.89%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P52</td>
<td>100%</td>
<td>100%</td>
<td>97.10%</td>
</tr>
<tr>
<td>P53</td>
<td>100%</td>
<td>100%</td>
<td>93.69%</td>
</tr>
<tr>
<td>P54</td>
<td>100%</td>
<td>100%</td>
<td>82.32%</td>
</tr>
<tr>
<td>P55</td>
<td>98.42%</td>
<td>97.80%</td>
<td>95.10%</td>
</tr>
<tr>
<td>P56</td>
<td>99.39%</td>
<td>99.22%</td>
<td>95.10%</td>
</tr>
<tr>
<td>P57</td>
<td>100%</td>
<td>100%</td>
<td>99.23%</td>
</tr>
<tr>
<td>P58</td>
<td>95.10%</td>
<td>100%</td>
<td>99.55%</td>
</tr>
<tr>
<td>P61</td>
<td>99.89%</td>
<td>100%</td>
<td>99.94%</td>
</tr>
<tr>
<td>P62</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P63</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P64</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>P65</td>
<td>98.00%</td>
<td>98.71%</td>
<td>99.78%</td>
</tr>
<tr>
<td>P66</td>
<td>99.94%</td>
<td>99.10%</td>
<td>97.24%</td>
</tr>
<tr>
<td>P67</td>
<td>100%</td>
<td>100%</td>
<td>94.94%</td>
</tr>
<tr>
<td>P68</td>
<td>99.97%</td>
<td>99.10%</td>
<td>94.94%</td>
</tr>
</tbody>
</table>

TABLE IV
CLASSIFYING BITMAP TEXTURES BY THREE APPROACHES

Fig. 3. Segmenting Two Textures with Multiple Regions

Fig. 4. Segmenting Two Textures with Complex Region Boundaries
of regularities they have captured. The understandability of GP programs is a difficult issue in this area. There is little research on this topic. The study on the behaviors of evolved classifiers could not only benefit our research of generating texture classifiers by GP, but also benefit GP in general.

The classifiers generated in the experiments described in previous sections are quite difficult to understand. To address this issue, two approaches are used to simplify classifiers: reduce the size of the problem and reduce the size of the classifiers themselves.

Following these approaches of simplifying classification, we carried out several investigations of classifiers and we discuss these throughout this section.

A. P14 against P24

Vertical lines (P14) and horizontal lines (P24) are the two simple textures. Enlarged $4 \times 4$ sub-images of these patterns are shown in Figure 5. We chose these as the starting point of our analysis. To begin, all functions and terminals are permitted to be used on these patterns. With this approach, GP can create a perfect classifier to differentiate P14 and P24, and it can often achieve this in the first generation. However, the generated classifiers are still difficult to understand. Therefore, to further simplify the problem, we performed a second set of experiments with these two patterns. The random number terminals and all the functions were disabled except for the “plus” and “minus” functions. With this approach, GP is only allowed to construct classifiers based on raw pixel inputs and the two simplest arithmetic operators. Additionally, a size penalty was applied to the fitness function:

$$f = \left( \frac{TP + TN}{TOTAL} \right)^{14} \times 100 - (ProgramSize)$$  \hspace{1cm} (2)

The classifiers in Functions 3 to 6 are simple: they sum values of 4 pixels selected from the possible 16 pixels. In these functions, $V_{a,b}$ is the intensity value of pixel $(a, b)$. The pixels that are selected by these classifiers are not arbitrarily chosen and there is an underlying regularity. Functions 3 and 4 select one pixel from each column of the $4 \times 4$ sub-image. However, the pixels selected are from only two rows. For texture P14—regardless of the position of the $4 \times 4$ sampling window—there is only one column that has all of the pixel values as 0 and the others are 1 as shown in Figure 5. Therefore, the output for classification of P14 using Functions 3 or 4 is always 3. However, for horizontal lines of P24 the output is 4—where no black line passes the pixels as in the second and the third P24 pattern in Figure 5—or 2 where one black line is picked as in the first and the fourth P24 pattern in Figure 5.

In contrast, Functions 5 and 6 select one pixel from each row of the input images, and the pixels are in pairs from two columns. The output for P24 sub-images is always 3, while that of P14 is either 2 or 4 as shown in Figure 6.

No programs use less than 4 pixels, which as can be seen from inspection, is the minimum requirement to distinguish these two patterns. Selected generated classifiers that use four pixels are shown below:

$$Output = V_{1,0} + V_{2,1} + V_{1,2} + V_{2,3}$$ \hspace{1cm} (3)

Class is P14, if $Output \in [3, 4]$.

$$Output = V_{2,0} + V_{3,1} + V_{2,2} + V_{3,3}$$ \hspace{1cm} (4)

Class is P24, if $Output < 3$ or $Output \geq 4$.

$$Output = V_{0,3} + V_{1,2} + V_{2,3} + V_{3,2}$$ \hspace{1cm} (5)

Class is P14, if $Output < 3$ or $Output \geq 4$.

$$Output = V_{0,3} + V_{1,2} + V_{2,0} + V_{3,0}$$ \hspace{1cm} (6)

Class is P24, if $Output \in [3, 4]$.

The other classifiers created that use only four pixels are simple variations that consider pixels in different positions.

B. P13 Against P23

Using the same techniques we investigated another two-class problem: classification of forward diagonals (P13) and backward diagonals (P23). Figure 7 shows the enlarged $4 \times 4$
sub-images of these two patterns. Around one-third of generated programs only use four pixels. Selected programs are listed below:

\[
\text{Output} = V_{0,0} + V_{1,0} + V_{1,1} + V_{2,1}
\]

(7)

**Class is P13, if Output ∈ [3, 4]**

**Class is P23, if Output < 3 or Output ≥ 4**

\[
\text{Output} = V_{0,2} + V_{0,3} + V_{1,3} + V_{3,2}
\]

(8)

**Class is P13, if Output ∈ [3, 4]**

**Class is P23, if Output < 3 or Output ≥ 4**

\[
\text{Output} = V_{0,0} + V_{1,0} + V_{1,3} + V_{2,3}
\]

(9)

**Class is P13, if Output ∈ [3, 4]**

**Class is P23, if Output < 3 or Output ≥ 4**

\[
\text{Output} = V_{2,3} + V_{2,2} + V_{3,2} + V_{3,1}
\]

(10)

**Class is P13, if Output < 3 or Output ≥ 4**

**Class is P23, if Output ∈ [3, 4]**

These classifiers also sum values of four selected pixels. The chosen pixels have a regular layout that is suitable for distinguishing the characteristics of the two patterns. For example, Function 7 uses pixels (0,0), (1,0), (1,1), and (2,1). Function 8 uses pixel (0,2), (0,3), (1,3), and (3,2). Functions 7 and 8 can be generalized as a four-pixel mask:

\[
\text{Output} = V_{y,x} + V_{y+1,x} + V_{y+1,x-1} + V_{y+2,x-1}
\]

(11)

**Class is P13, if Output ∈ [3, 4]**

**Class is P23, if Output < 3 or Output ≥ 4**

Wherever we place this mask on P13, there is only one pixel with the value 0 as shown in Figure 8. Therefore, the outputs for P13 sub-images are always 3. However, when masking the backward diagonal P23, there are either two or no black pixels captured by this mask.

Accordingly, the classifiers shown in Functions 9, 10, and ?? can be generalized as:

\[
\text{Output} = V_{y,x} + V_{y+1,x} + V_{y+1,x-1} + V_{y+2,x-1}
\]

(12)

**Class is P13, if Output < 3 or Output ≥ 4**

**Class is P23, if Output ∈ [3, 4]**

\[
\text{Output} = V_{y,x} + V_{y-1,x-1} + V_{y-2,x-1}
\]

(13)

**Class is P13, if Output < 3 or Output ≥ 4**

**Class is P23, if Output ∈ [3, 4]**

\[
\text{Output} = V_{y,x} + V_{y+1,x+1} + V_{y+1,x+1} + V_{y+1,x+2}
\]

(14)

**Class is P13, if Output ∈ [3, 4]**

**Class is P23, if Output < 3 or Output ≥ 4**

To classify P13 and P23, GP creates a set of Z-shaped masks to detect the regularity of the patterns. Although these masks have different orientations—as shown in Figure 9—the masks are small enough to be understood and sufficient to distinguish between these simple textures.

**C. P41 against P14, P24, P13 and P23**

We extend here our investigation to an analysis of the classification of a mixture of five patterns: grids against lines and diagonals. Figure 10 shows the enlarged 4 × 4 sub-images of P41. After 50 generations, the smallest programs found again use 4 pixels, while others use 5 or 6 pixels. Two examples of the small classifiers are shown below:

\[
\text{Output} = V_{0,1} + V_{1,1} + V_{2,1} + V_{3,1}
\]

(15)

**Class is P41, if Output ∈ [2, 3]**

**Other classes, if Output < 2 or Output ≥ 3**

\[
\text{Output} = V_{0,3} + V_{1,3} + V_{2,3} + V_{3,3}
\]

(16)

**Class is P41, if Output ∈ [2, 3]**

**Other classes, if Output < 2 or Output ≥ 3**
Such a layout is quite straightforward: the classifier actually calculates the frequency of pixel value 1 within a single column. P41 always has two 1s in each column. However, for other patterns, the numbers of pixels that are set as 1 is either 1 or 4—a vertical line—but never 2. Therefore, by looking at a column, the classifiers can correctly classify P41 as distinct from mixed pattern images.

VI. CONCLUSION

The aim of this study is to explore the applicability of genetic programming in the field of texture analysis. For the tasks of texture classification, this study illustrates that GP can achieve high accuracy in classifying texture features. Compared to conventional classification methods such as C4.5, GP generated classifiers are good although the feature vectors have a large number of attributes.

The further study in texture classification shows the feasibility of the single-step approach. That is, texture classifiers can be evolved directly based on raw-pixels, without the conventional feature extraction phrase. So by using GP, a new paradigm of texture classification can be established.

Furthermore, our study shows that GP evolved classifiers can be extended to a more complex task, texture segmentation. By adapting the classifiers generated from the single-step approach, a method can be developed to achieve fast and accurate texture segmentation. This method can produce relatively smooth boundaries and handle complex boundaries.

By analyzing generated classifiers, it can been seen that the success of GP classifiers is not ad-hoc. There are regularities of textures been captured. For the simple tasks, the classifiers behave as template matchers and frequency analyzers. For the more complex problems - such as classifying one texture against large group of other textures and classifying grey level textures - the generated classifiers are more difficult to understand. However we believe that certain underlying regularities are also being captured.

In a word, genetic programming can be used in the complex domain, texture analysis, either texture classification or texture segmentation. Good performance can be achieved by GP approaches. In our future work, there are a large number of topics worth pursuing such as optimal choice of function set, terminal set; optimal run-time parameters; multi-class texture classification and more investigation on the understandability of GP-generated texture classifiers.

REFERENCES