Evolutionary Image Descriptor: A Dynamic Genetic Programming Representation for Feature Extraction

Harith Al-Sahaf[1] Mengjie Zhang[1] Mark Johnston[2]
[1]School of Engineering and Computer Science
[2]School of Mathematics, Statistics and Operations Research
Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
{harith.al-sahaf, mengjie.zhang}@ecs.vuw.ac.nz, mark.johnston@msor.vuw.ac.nz

ABSTRACT

Texture classification aims at categorising instances that have a similar repetitive pattern. In computer vision, texture classification represents a fundamental element in a wide variety of applications, which can be performed by detecting texture primitives of the different classes. Using image descriptors to detect prominent features has been widely adopted in computer vision. Building an effective descriptor becomes more challenging when there are only a few labelled instances. This paper proposes a new Genetic Programming (GP) representation for evolving an image descriptor that operates directly on the raw pixel values and uses only two instances per class. The new method synthesises a set of mathematical formulas that are used to generate the feature vector, and the classification is then performed using a simple instance-based classifier. Determining the length of the feature vector is automatically handled by the new method. Two GP and nine well-known non-GP methods are compared on two texture image data sets for texture classification in order to test the effectiveness of the proposed method. The proposed method is also compared to three hand-crafted descriptors namely domain-independent features, local binary patterns, and Haralick texture features. The results show that the proposed method has superior performance over the competitive methods.

CCS Concepts

•**Computing methodologies** → **Genetic programming;** *Interest point and salient region detections; Matching;* Feature selection;

Keywords

Genetic Programming, Multiclass classification, Textures

1. INTRODUCTION

Texture analysis can be divided into at least four categories that each aim at investigating a specific task [25]. In *texture synthesis*, the aim is to build a model that can be used to construct the texture. *Texture segmentation* aims at dividing an image into a number of homogeneous regions based on their texture properties. In *texture classification*, the goal is to assign different class labels for instances of different classes. Finally, *shape extraction from texture* aims at exploiting texture information to construct a 3D surface projection from the 2D surface geometry.

A large number of methods for texture analysis have been proposed over the past few decades. Tuceryan and Jain [25] divided these methods into four approaches: (1) structural; (2) statistical; (3) transform; and (4) model-based. The methods in each of these four approaches perform texture analysis differently. For example, structural methods attempt to understand the underlying texture relying on well-known primitives and their arrangements; whereas statistical methods rely on the distribution of the gray levels to indirectly represent the texture using non-deterministic properties. More details can be found in [25].

Genetic Programming (GP) is a biologically inspired evolutionary computation technique that replicates the Darwinian principles of natural selection and survival of the fittest [10]. GP starts from a population of randomly generated programs, each of which represents a solution, then gradually evolves these solutions over a number of generations via using genetic operators to explore the solution space [20].

Over the past few decades, utilising GP for image related problems has attracted increasing interest [19, 1]. Two GP methods have been developed by Song et al. [21] for multiclass texture classification by utilising the *Static Range Selection* (SRS) [22, 30] and *Dynamic Range Selection* (DRS) [13] techniques. In [24] a GP methodology to detect interest points on images by synthesising low-level image operators has been described. In [18], Olague et al. extended the work of [24] by presenting an innovative GP method for evolving an interest points detector. Synthesising mathematical formulas using GP as a preprocessing step to improve the performance of the *Scale-invariant Feature Transform* (SIFT) descriptor [14] is proposed in [19]. On the contrary, Hindmarsh et al. [7] used GP as a postprocessing step to select a subset of the features detected by SIFT for object recognition. Recently, Albukhanajer et al. [1] adopted a multi-objective approach in conjunction with the *trace transform* to extract image features that are robust to noise and invariant to different geometric deformations.

Most of the aforementioned methods require either a large number of instances to evolve a model, and/or human inter-

GECCO '15, July 11 - 15, 2015, Madrid, Spain

DOI: http://dx.doi.org/10.1145/2739480.2754661

vention (mostly domain experts) to design and extract a good set of features.

Conventionally, a program evolved by GP for *binary* classification produces a single value from the root node for each instance, which is then translated to a class label using the zero value to divide the real number line into two intervals [20]. Thus, the instance being evaluated is considered as belonging to one class (e.g. foreground) if the value resulting from the root node is negative; and belonging to the other class (e.g. background) if it is positive or zero.

Utilising GP for *multiclass* classification can be divided into at least four groups. Output translation is the first group, where the real number line is split into intervals each of which corresponds to one class, and the output of the evolved program is translated to a class label based on the interval it falls in [30]. Wrapper-based methods represent the second group, where a classifier is used with the program evolved by GP to predict the class label [29]. In the wrapper approach, the evolved program performs feature selection or extraction and those features are fed into the wrapped classifier to perform classification. Methods of the third group aim at producing multiple values from the evolved program, which require modifying the program representation [32]. In the fourth group, the problem of multiclass classification is decomposed into a number of binary classification tasks [13]. The second approach, i.e., wrapper, is adopted in this study where a simple instances-based classifier is used namely, *k-Nearest-Neighbour* (kNN) with k is set to 1 (1NN).

Goals

This paper aims at developing a new GP method for texture primitives detection. The new method performs two essential tasks. Firstly, automatically evolving a descriptor that is capable of detecting and extracting texture features. Secondly, performing multiclass texture classification using the features generated by the evolved descriptor and a simple classifier (1NN). Precisely, this paper aims at addressing the following research objectives:

- develop an appropriate individual representation and fitness function that allows GP to automatically detect and extract image features;
- investigate whether the proposed method is capable of outperforming other GP-based methods that are designed to operate on raw pixel values for multiclass texture classification; and
- investigate whether the evolved descriptor can compete with well-known descriptors that are designed by domain experts using nine widely used classifiers.

The rest of the paper is organised as follows. Three image feature extraction methods are briefly discussed in Section 2. Section 3 describes the proposed GP approach for evolving feature descriptors. The experimental setup, parameter settings, data sets, and baseline methods are presented in Section 4. Section 5 presents and discusses the results along with two evolved descriptors. Section 6 concludes this paper and highlights some future work directions.

2. RELATED WORK

This section describes two descriptors, namely *local binary patterns* [17] and *Haralick texture features* [6], and a domain-independent image feature extraction method [31].

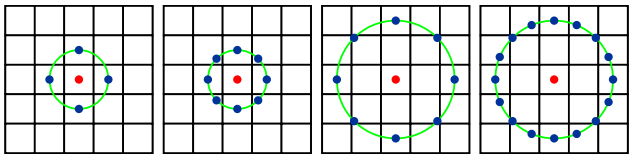


Figure 1: Examples demonstrate different settings of LBP by changing the value of n and r. From left to right, $LBP_{4,1}$, $LBP_{8,1}$, $LBP_{8,2}$, and $LBP_{16,2}$.

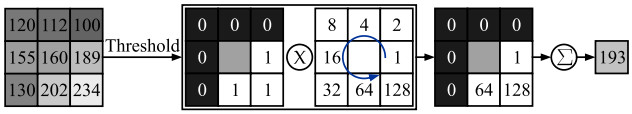


Figure 2: Example shows the required steps to generate a binary code, then convert the generated code to decimal value.

2.1 Local Binary Patterns

In 1994, Ojala et al. [17] proposed a new texture image descriptor called Local Binary Patterns (LBP). The main aim of LBP is to generate a binary code at each pixel relying on the values of the neighbouring pixels, and then use the generated codes to compute a histogram (i.e. feature vector). The process of computing the histogram consists of four steps. Firstly, a histogram of size 2^L bins is constructed and initialised, where L is the length of the binary code. The length of the binary code is defined based on two parameters: (1) the number of neighbouring pixels n; and (2) the distance between the central pixel and each of the neighbouring pixels, denoted as the radius r. LBP and these two parameters are denoted as $LBP_{n,r}$. Figure 1 shows different settings of n and r. Originally, LBP operated using a sliding window of size 3×3 pixels ($LBP_{8,1}$). Secondly, the image is scanned in a pixel-by-pixel fashion and each of the neighbouring pixels surrounding the current central pixel of the sliding window is set to 0 if its value is less than that of the central pixel; otherwise, it is set to 1 (as presented in Figure 2). Thirdly, these values are used to form a binary code, and then multiplied by powers of 2 in an anticlockwise direction to convert the binary code to a decimal value. Fourthly, the bin corresponding to the generated decimal value in the third step is incremented by 1.

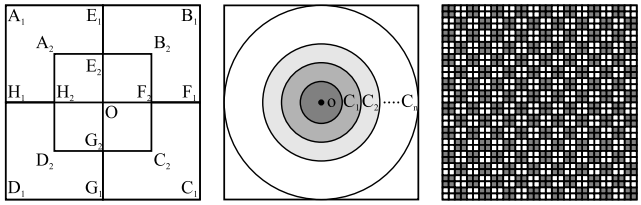
2.2 Domain Independent Features (DIFs)

Zhang et al. [31] proposed three domain-independent terminal sets to extract image features: (1) rectilinear features; (2) circular features; and (3) pixel features. These methods use statistical properties, i.e., mean and standard deviation, of predefined image regions to extract the feature vector. Figure 3 highlights the regions of these three methods.

The rectilinear method is used in this study as a competitive method for feature extraction. A feature vector generated by the rectilinear method consists of 20 features that are extracted from 10 regions as listed in Table 1. This method combines both *global* (i.e. the entire image) and *local* (i.e. regions of the image) features, which has been shown to be more powerful than use only one of them [12].

2.3 Haralick Texture Features

Haralick Texture Features [6] is a statistical texture analysis method that uses the *Gray-Level Co-occurrence Matrix*



(a) (b) (c)

Figure 3: The regions of the (a) rectilinear, (b) circular, and (c) pixels features [31].

Table 1: Pixel statistics of the rectilinear method.

Features		Regions of interest	Features		Regions of interest
μ	σ		μ	σ	
F_1	F_2	Square $A_1B_1C_1D_1$	F_{11}	F_{12}	Square $A_2B_2C_2D_2$
F_3	F_4	Quadrant $A_1E_1OH_1$	F_{13}	F_{14}	Horizontal line H_1F_1
F_5	F_6	Quadrant $E_1B_1F_1O$	F_{15}	F_{16}	Horizontal line H_2F_2
F_7	F_8	Quadrant $H_1OG_1D_1$	F_{17}	F_{18}	Vertical line E_1G_1
F_9	F_{10}	Quadrant $OF_1C_1G_1$	F_{19}	F_{20}	Vertical line E_2G_2

(GLCM) to extract texture features [25]. The process of extracting features from an image starts by generating a set of co-occurrence matrices each of which is generated by counting the number of adjacent pixels of a predefined offset and direction (i.e. angle). Then a number of features are calculated from each matrix to form the feature vector.

In this study, for each instance, four matrices are generated using a single pixel offset and the directions 0°, 45°, 90°, and 135°. The feature vector is then generated by calculating from each of the matrices the *Homogeneity* (inverse difference), *Dissimilarity* (absolute value), *Contrast* (inertia), *Energy* (angular second moment), *Entropy*, and *Correlation* that are formally defined as:

$$Homogeneity = \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{p\,(i,j)}{1+|i-j|} \tag{1}$$

$$Dissimilarity = \sum_{i=1}^{H}\sum_{j=1}^{W} |i-j|\,p\,(i,j) \tag{2}$$

$$Contrast = \sum_{i=1}^{H}\sum_{j=1}^{W} (i-j)^2\,p\,(i,j) \tag{3}$$

$$Energy = \sum_{i=1}^{H}\sum_{j=1}^{W} p\,(i,j)^2 \tag{4}$$

$$Entropy = \sum_{i=1}^{H}\sum_{j=1}^{W} p\,(i,j)\,[-\log_2 p\,(i,j)] \tag{5}$$

$$Correlation = \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{(i-\mu_i)\,(j-\mu_j)\,p\,(i,j)}{\sigma_i\sigma_j} \tag{6}$$

where H and W are, respectively, the height and width of the matrix, $p\,(i,j)$ is the value at the i^{th} row and j^{th} column, and $|\cdot|$ returns the absolute value of the argument. The μ_i, μ_j, σ_i and σ_j are, respectively, the mean and standard deviation of the i^{th} row and j^{th} column.

3. THE NEW METHOD

This section discusses the evaluation procedure, program representation, feature vector extraction, and fitness function of the proposed *Evolutionary Image Descriptor* (EID).

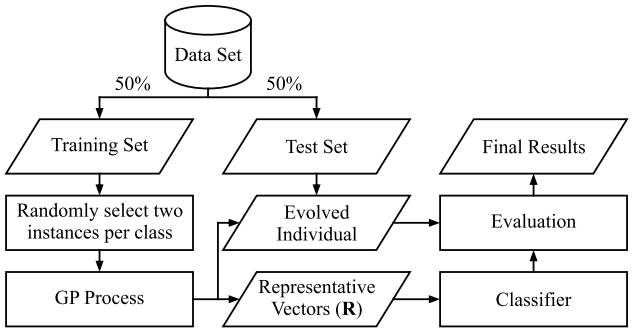


Figure 4: Flowchart shows the evaluation procedure.

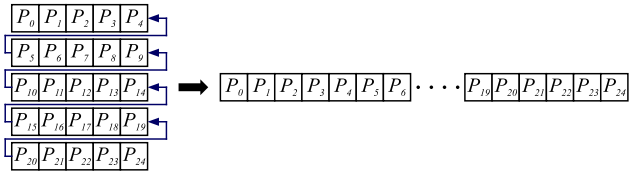


Figure 5: Converting a window of size 5×5 pixels to a 1D vector.

3.1 The Evaluation Procedure

The evaluation procedure of EID consists of the typical machine learning *training* and *testing* phases. As presented in Figure 4, the process starts by dividing the content of the data set equally between the training and test sets. The system randomly selects only two instances per class from the training set and feeds them to the GP process. The GP process runs until a stopping criterion is met. The results of the GP process are: (1) the best evolved program; and (2) a set of representative vectors denoted as **R**. The system then uses the evolved program to generate the feature vectors of the test set instances, and uses **R** as the knowledge base for a 1NN classifier to predict the class label of each instance. These steps are discussed more in the following subsections.

Only two instances of each class are randomly selected from the training set to evolve a program. Using different instances can give different results. Thus, the process depicted in Figure 4, apart from the step of dividing the data set into training and test sets, has been further repeated 30 times.

3.2 Program Representation

In this paper, the tree-based GP [10] program structure is used to represent an individual evolved by the proposed method. Moreover, *Strongly-typed GP* (STGP) is used to define constraints on the order of the different nodes in the program tree and to preserve the closure property [16].

The *terminal set* consists of the raw pixel values of a sliding window with a predefined size. In our experiment, the window size is set to 5×5 pixels. The pixels of each window are converted to a 1D vector as demonstrated in Figure 5. Thus, the terminal nodes are randomly selected from $\{P_0, P_1, \ldots, P_{24}\}$. The windows that exceed the boundaries of the instance being evaluated are discarded. The pixel with the coordinates $(0, 0)$ represents the upper left corner of the image; whereas the pixel at the lower right corner of the image has the coordinates $(M-1, N-1)$ where M and N are, respectively, the width and height of the image.

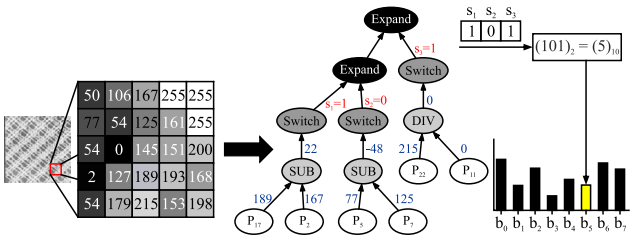


Figure 6: The steps required to generate the feature vector.

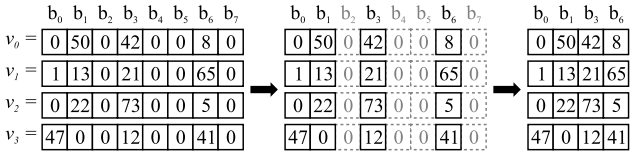


Figure 7: The steps to remove the unwanted bins from the extracted vectors v_0, v_1, v_2, and v_3.

The *function set* is made up of the four standard arithmetic operators $+$, $-$, $\times$, and $\div$, in addition to the *Switch* and *Expand* nodes as shown in Figure 6. Each of the four mathematical operators has its regular meaning that applies the corresponding operation on two arguments and returns a single floating-point value. However, the $\div$ operator returns 0 if the divisor is zero. The input and output of these four operators are of the same type; thus, they can be used to form a chain of operators. The *Switch* node takes only one argument and returns 0 if the input is negative, and 1 otherwise. Each *Switch* node represents a single bit of the code generated at each position of the sliding window (more details in the following subsection). Meanwhile, the *Expand* node takes two arguments, each of which can be *Switch* or *Expand* nodes; the *Expand* node is used to allow GP to evolve trees of different sizes.

3.3 Extracting the Feature Vector

The process of extracting the feature vector from an instance requires to iterate over the pixel values of the instance using a sliding window of a predefined size. The process starts by generating an empty vector of length 2^b, where b is the total number of *Switch* nodes in the tree. Then at each position of the sliding window, the system performs two steps as shown in Figure 6. Firstly, the system feeds the values of the current window to the terminal nodes, performs the arithmetic operations, and generates a binary code using the values returned by the *Switch* nodes. Secondly, the generated code is converted to decimal and the bin at the corresponding index of the vector is incremented by 1.

The resulting vectors can be too long due to having a large number of *Switch* nodes. Reserving large chunks of the machine memory can result in occupying the entire memory, which will lead to immediate termination. To prevent this situation, a simple analysis of the resulted vectors has been performed and show that many of these bins have zeros. Thus, these bins that have 0 across all vectors have been removed and reported as *unwanted* as shown in Figure 7.

3.4 Generating the Representative Vectors

The proposed method generates a single vector per class using the extracted feature vectors from the training set instances. The representative vector of a class is generated by averaging the values of each bin across all feature vectors of that class as presented in Figure 8. The generated representative vectors are stored in a set denoted as $\mathbf{R}$ that is used to perform two tasks, firstly, to help measure the fitness value (more details in the following subsection), and secondly, to serve as the knowledge base to classify the instances of the unseen data (i.e. test set). For each of the unseen instances, the system generates the feature vector, removes the previously indicated unwanted bins, calculates the distance between the instance being evaluated and every representative vector (Equation (10)), and returns the class label of the closest representative vector (1NN).

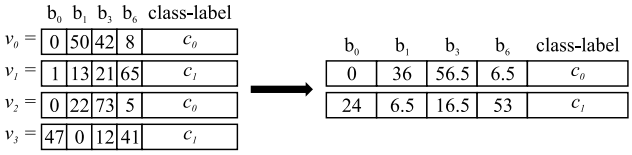


Figure 8: Generating the representative vectors of classes c_0 and c_1.

3.5 Fitness Function

Operating directly on the raw pixel values and using only two instances per class, are two essential components of EID. Thus, relying on the typical standard accuracy of the training set is inadequate due to having a large feature space and a few training instances. A compound fitness function is used in order to tackle both of these issues as shown in Equation (7).

$$Fitness = 1 - \left(\frac{Accuracy + Distance_{\mu}}{2} \right) \tag{7}$$

Here, *Accuracy* has its regular meaning, that is the proportion of correctly classified instances (N_{correct}) to the total number of instances (N_{total}) as in Equation (8).

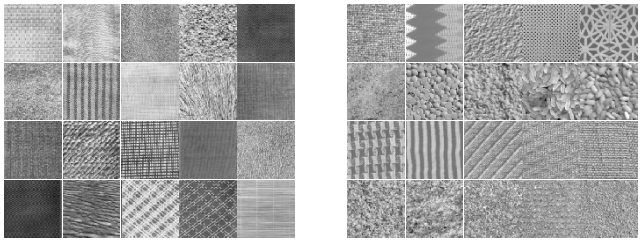
$$Accuracy = \frac{N_{\text{correct}}}{N_{\text{total}}} \tag{8}$$

The accuracy measures the ability of the evolved program to correctly classify the instances of the training set. To calculate the accuracy, the system iterates over the instances of the training set and for each, the class label is predicted using the representative vectors as the knowledge base with 1NN. Then a `hit` is returned if the predicted label matches the actual label; otherwise, a `miss` is returned.

The proposed method also aims to generate a distinctive feature vector for instances in the same class compared to those instances of the other classes, which is handled by the second component of the fitness function, i.e., $Distance_{\mu}$. The $Distance_{\mu}$ component is the average distance between each representative vector of one class and the closest representative vector of other classes as presented in Equation (9).

$$Distance_{\mu} = \frac{1}{C} \sum_{\mathbf{v} \in \mathbf{R}} \underset{\{\mathbf{u} \in \mathbf{R} \setminus \mathbf{v}\}}{\operatorname{argmin}} D(\mathbf{v}, \mathbf{u}) \tag{9}$$

Here, C is the total number of classes, $\mathbf{R}$ is the set of representative vectors, and $\mathbf{v}$ and $\mathbf{u}$ are two vectors in $\mathbf{R}$. The $D(\cdot, \cdot)$ function measures the distance between two vectors, which is calculated using *Czekanowski Coefficient* [2] that is



(a) (b)

Figure 9: Samples of (a) DS-Brodatz that are from left to right and top to bottom D1, D3, D4, D5, D6, D9, D11, D14, D15, D16, D17, D18, D20, D21, D24, D34, D37, D46, D47, and D49; and (b) DS-Kylberg that are from left to right and top to bottom blanket1, blanket2, ceiling1, ceiling2, floor1, floor2, lentils1, pearl-sugar1, rice1, rice2, scraf1, scarf2, screen1, seat1, seat2, stone1, stone2, stone3, stone-slab1, and wall1.

formally defined as:

$$D(\mathbf{v}, \mathbf{u}) = 1 - \left(\frac{2\left(\sum_{v \in \mathbf{v}, u \in \mathbf{u}} \min(v, u)\right)}{\sum_{v \in \mathbf{v}} v + \sum_{u \in \mathbf{u}} u} \right) \quad (10)$$

where v and u are two corresponding elements in the vectors $\mathbf{v}$ and $\mathbf{u}$ respectively. Meanwhile, the $\min(\cdot, \cdot)$ function returns the minimum value of the two arguments.

4. EXPERIMENT SETUP

The data sets, instance preparation, baseline methods, and evolutionary parameters are outlined in this section.

4.1 Data Sets and Preparation

Two widely used, texture based, gray-scale data sets are used in this study to test the proposed method. The instances of the first data set are selected from the *Brodatz Texture* data set [3], and denoted as *DS-Brodatz* in this study. The original Brodatz data set consists of 112 classes, each consists of a single instance with size 640×640 pixels. Only 20 out of the 112 classes have been randomly drawn to form DS-Brodatz. Figure 9(a) shows samples and names of the selected classes.

The second data set (DS-Kylberg) is formed using 20 randomly selected classes of the *Kylberg Texture* data set [11] as shown in Figure 9(b). The Kylberg data set consists of 28 classes that each comes in *with* and *without* rotation. Only the latter type is considered in this study as extending the proposed method to handle the rotation variant will be investigated in the future. Each class of this group consists of 160 instances, each with size 576×576 pixels.

The instances of the DS-Kylberg data set have been resized to be of size 115×115 pixels each, in order to reduce the computation costs. Meanwhile, the single instance in each class of DS-Brodatz has been re-sampled by dividing the image into 100 sub-images with size 64×64 pixels each. The total number of instances has been equally split into training and test sets. Thus, each of the training and test sets of DS-Brodatz consists of $1,000$ instances (50 instances $\times$ 20 classes); whereas each of the training and test sets of DS-Kylberg is made up of $1,600$ instances (80 instances $\times$ 20 classes). Only two instances of each class are randomly selected to form the training set and the rest of the available pool is discarded; while the test set is kept identical in the entire experiment. The instances of both DS-Brodatz and DS-Kylberg have been standardised and normalised as preprocessing steps in order to eliminate or reduce the effect of illumination variation. Equation (11) is used for standardisation. The resulting instance after standardisation has zero mean and unit standard deviation. Then to normalise the standardised instance, Equation (12) is used. The pixel values after normalisation are in the interval $[0, 255]$.

$$x' = \frac{x - \bar{x}}{s} \quad (11)$$

$$x' = \left(\frac{x - x_{\min}}{x_{\max} - x_{\min}} \right) \times 255 \quad (12)$$

Here, x' and x are, respectively, the new and old pixel values. The mean and standard deviation are denoted by $\bar{x}$ and s, and $x_{\min}$ and $x_{\max}$ are the minimum and maximum values respectively.

4.2 Methods for Comparison

The proposed method is compared to 11 classifiers of different types. The competitive methods can be categorised into two groups: (1) GP-based; and (2) non-GP. The former group consists of the SRS and DRS methods, whilst the latter group consists of Naïve Bayes (NB), Support Vector Machines (SVM), Decision Trees (J48), hybridised Naïve Bayes/Decision Trees (NBTree), K* (KStar), Non-nested generalised (NNge), Adaptive Boosting M1 (ABM1), Multilayer Perceptron (MLP), and Random Forest (RF). We have reimplemented both of the SRS and DRS methods as in [21] using the platform provided by the *Evolutionary Computation Java-based* (ECJ) package [15]. Meanwhile, the implementation of the other methods have been taken from the *Waikato Environment for Knowledge Analysis* (WEKA) package [5] that is discussed in detail in [28].

4.3 Parameter Settings

The parameter settings of both GP and non-GP methods are highlighted here.

4.3.1 GP Evolutionary Parameters

The parameters of the three GP methods, i.e., SRS, DRS, and EID, are summarised in Table 2 and kept identical in the experiment to make fair comparisons. These parameters are based on previous work in the literature. The population of 200 individuals is generated using the ramped-half-and-half method [10]. The probabilities of the elitism, mutation, and crossover operators are, respectively, 0.01, 0.19, and 0.80. Selecting individuals to participate in generating the subsequent generation uses the tournament method with size 7. The minimum tree depth is set to 2, whereas the maximum depth is set to 10 to allow GP to evolve large trees, but not too large in order to avoid the *code bloating* problem [26]. The GP process is terminated after 50 generations if no *ideal* solution is found.

4.3.2 Non-GP Parameters

Some optimisation and parameter tuning have been applied to the non-GP methods. The number of neighbours in all instance-based methods such as NNge and KStar has been set to 1 due to having only two instances per class.

Table 2: The GP evolutionary parameters

Parameter	Value	Parameter	Value
Elitism Rate	0.01	Generations	50
Mutation Rate	0.19	Population Size	200
Crossover Rate	0.80	Selection Type	Tournament
Minimum Depth	2	Tournament size	7
Maximum Depth	10	Initial Population	Half-and-half

The structure of the MLP classification method has been set based on the guidelines proposed by Trenn [23], where the number of hidden neurons (N_{hidden}) is calculated as:

$$N_{\text{hidden}} = \left\lceil \frac{N_{\text{in}} + N_{\text{out}}}{2} \right\rceil \quad (13)$$

where N_{in} and N_{out} are, respectively, the number of neurons in the input and output layers.

A study by Keerthi and Lin [9] showed that SVM with a non-linear kernel has the potential to achieve comparable or better performance than that with a linear kernel. Therefore, SVM is used in this study with a Gaussian kernel instead of the default linear kernel of WEKA. Different classifiers have been tested with ABM1, and the best results have been achieved when LADTree [8] is used.

5. RESULTS AND DISCUSSIONS

The results obtained from the experiments are presented and discussed in this section.

5.1 Overall Results

The aim of the conducted experiments is to examine the performance of the proposed EID method. To this end, an experiment has been designed using the DS-Brodatz and DS-Kylberg data sets. The results of the experiment on each data set are presented in a table that vertically consists of two parts. The results of the three GP methods (SRS, DRS and EID) occupy the first (upper) part, whereas non-GP methods occupy the second (lower) part. Moreover, the second part horizontally is made up of three columns that each presents the performances of different classifiers using the features extracted by DIF, LBP, and GLCM. The *Wilcoxon signed-ranks* test [27, 4] with a significance level of 5% is used to identify whether the performance of EID is significantly different than that of the other methods, which is indicated using a "$*$" symbol.

Table 3 presents the results obtained on the DS-Brodatz data set. The proposed method has significantly outperformed both of the GP baseline methods. Compared to non-GP methods with three sets of hand-crafted features, EID has achieved significantly better performance.

The results on the DS-Kylberg data set are presented in Table 4. Similar to DS-Brodatz, the significance test shows that the gap between the performance of the proposed method and that of SRS and DRS is significant. Moreover, non-GP methods have performed significantly worse than the proposed method on this data set.

5.2 Further Discussion

In order to get better understanding of the structure of the program evolved by EID, one of the best programs evolved on each data set is selected for further discussion.

Figure 10 depicts the tree representation of a program evolved by EID on the DS-Brodatz data set. This program has achieved 95.80% accuracy on the unseen data. The program generates a feature vector of length 2^6 bins (assuming no unwanted bins need to be removed) due to having 6 *Switch* nodes. Thus, a binary code consists of 6 bits is generated for each position of the sliding window. Four out of those 6 bits are generated by simply subtracting the values of the arguments, whilst the other two require an additional operator to be applied. This reflects the simplicity and low-cost of the evolved program, and more importantly, it can be used for online applications. The use of the subtraction operator more than the other operators was expected due to the influence of this operator on flipping the sign of the resulting value (i.e. 0 to 1 and vice versa).

Table 3: The accuracies (%) on the test set for DS-Brodatz ($\bar{x} \pm \sigma$).

	SRS	DRS	EID
	4.88 ± 0.14*	10.23 ± 0.61*	95.39 ± 0.90
	DIF	LBP	GLCM
NB	25.36 ± 5.86 *	67.22 ± 6.17 *	62.40 ± 8.18 *
SVM	39.58 ± 4.07 *	71.66 ± 4.93 *	75.94 ± 3.39 *
NBTree	42.77 ± 4.35 *	77.66 ± 4.38 *	64.33 ± 5.45 *
KStar	45.32 ± 3.19 *	79.26 ± 2.38 *	76.62 ± 4.97 *
NNge	41.60 ± 4.13 *	80.40 ± 3.48 *	78.71 ± 4.89 *
MLP	40.88 ± 2.73 *	84.10 ± 2.69 *	80.97 ± 4.59 *
ABM1	17.63 ± 4.64 *	56.00 ± 15.0 *	22.52 ± 6.27 *
J48	33.96 ± 5.09 *	32.41 ± 6.47 *	47.33 ± 8.11 *
RF	41.90 ± 3.00 *	63.99 ± 1.21 *	65.40 ± 3.80 *

Table 4: The accuracies (%) on the test set for DS-Kylberg ($\bar{x} \pm \sigma$).

	SRS	DRS	EID
	5.09 ± 0.18*	8.83 ± 0.53*	92.01 ± 1.02
	DIF	LBP	GLCM
NB	24.96 ± 4.76 *	76.22 ± 6.74 *	63.58 ± 5.61 *
SVM	27.54 ± 2.74 *	77.72 ± 4.51 *	79.80 ± 4.55 *
NBTree	30.14 ± 4.10 *	77.71 ± 5.93 *	65.96 ± 5.34 *
KStar	24.81 ± 2.17 *	87.09 ± 2.14 *	80.50 ± 5.39 *
NNge	34.40 ± 3.24 *	87.80 ± 2.54 *	80.32 ± 5.34 *
MLP	29.29 ± 2.83 *	85.83 ± 2.63 *	81.46 ± 4.55 *
ABM1	15.69 ± 4.24 *	54.91 ± 10.4 *	22.64 ± 5.32 *
J48	35.27 ± 4.41 *	35.89 ± 4.34 *	50.75 ± 5.22 *
RF	29.19 ± 1.81 *	68.48 ± 1.77 *	66.85 ± 3.89 *

The tree representation of a program evolved by the proposed method on DS-Kylberg is depicted in Figure 11. On the unseen data, this program showed 96.94% accuracy. A closer inspection reveals that the feature vector generated for an instance by this program is of length 2^5 bins. Moreover, at most 4 operators are required to calculate the value of each bit of the binary code generated at each window position. Similarly, the subtraction operator appears more frequently than the other operators.

5.3 Discussion on Number of Instances

Another experiment is conducted in this study in order to investigate the effect of having a larger training set on the performance of the baseline methods. Thus, all the instances of the training set are used to train or evolve a model instead of selecting only two instances per class.

The results of the two GP baseline methods are presented in Table 5, which show that increasing the number of instances has improved the performance of SRS and DRS. However, these two methods still show very poor performances compared to that of EID using only two instances

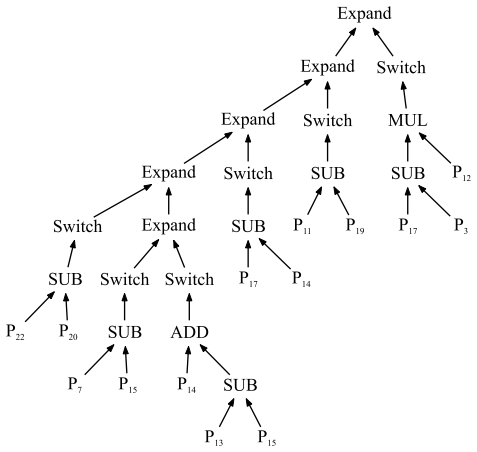


Figure 10: A program evolved on DS-Brodatz.

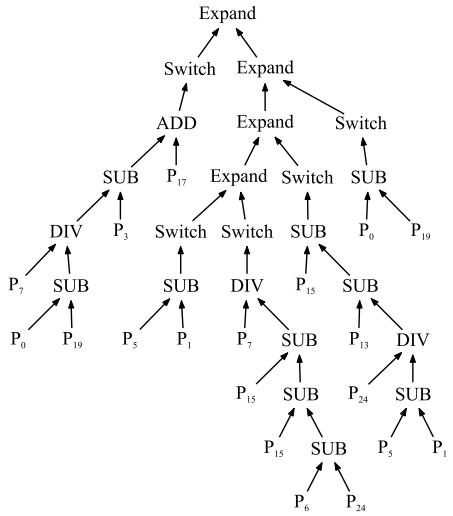


Figure 11: A program evolved on DS-Kylberg.

per class. Although SRS and DRS have been designed to tackle the multiclass classification problem and have been shown to achieve a high level of performance in [21], it requires further investigation to identify whether the number of classes or the size of those instances (e.g. Song et al. [21] used instances of size 16×16 pixels; whilst we are using instances with is 64×64 pixels) affects the performance.

Table 6 presents the results of all non-GP methods on the two data sets using the entire training set. The deterministic methods have been executed only one time and the achieved performance on the unseen data is reported in the table. Meanwhile, the average performance over 30 independent runs is reported for the non-deterministic methods (e.g. MLP). The obtained results show that increasing the number of training instances has the potential to improve the performances. Clearly, using all the instances in the training set increases the performances of all these commonly used machine learning methods over using only two instances per class. However, the new EID method using only two instances per class still achieved better (or comparable) results than most of these commonly used machine learning methods using a large number of instances in the training set. In the future, we will investigate why EID can perform so well using only two instances per class.

Table 5: The accuracies (%) of the GP baseline methods on the test set ($\bar{x} \pm \sigma$).

DS-Brodatz		DS-Kylberg	
SRS	DRS	SRS	DRS
6.73 ± 1.31	11.58 ± 1.21	7.35 ± 1.07	9.60 ± 0.74

Table 6: The accuracies (%) of the non-GP baseline methods on the test set.

	DS-Brodatz			DS-Kylberg		
	DIF	LBP	GLCM	DIF	LBP	GLCM
NB	72.20	92.70	88.40	70.75	95.63	91.00
SVM	50.00	67.00	77.20	38.81	76.63	85.13
NBTree	60.20	84.00	86.30	60.06	92.19	89.00
KStar	68.50	95.90	95.60	45.00	98.44	97.94
NNge	64.90	87.40	92.00	58.38	94.63	93.13
MLP	68.57	97.27	97.76	62.78	95.57	97.49
ABM1	41.80	92.80	33.10	44.69	53.63	63.69
J48	63.30	85.00	85.40	64.06	89.75	89.75
RF	68.65	94.25	91.28	65.55	95.77	94.38

6. CONCLUSIONS

The overall aim of this paper was to utilise GP for evolving a program that operates on the raw pixel values and capable of detecting texture primitives using only two instances per class. This goal has been successfully achieved as the program evolved by the proposed system generates the feature vector from the instance being evaluated using a sliding window, and predicts the class label relying on a knowledge base formed using the instances of the training set. Unlike hand-crafted image descriptors, the developed system automatically selects the length of the feature vector and the required formulas to generate this vector. To investigate the effectiveness of the proposed method, an experiment has been designed using two well-known and publicly available data sets namely, *Brodatz Texture* and *Kylberg Texture*. The experiment aims at investigating the ability of the developed system to discriminate between instances of different textures. The performance of the new method has been compared to two GP and nine non-GP widely used methods namely, NB, SVM, NBTree, KStar, NNge, MLP, ABM1, J48, and RF using three well-known hand-crafted feature extraction methods. The results suggest that the new method has significantly outperformed all comparative methods in terms of classification performance on both data sets and over the use of hand-crafted features.

The robustness of the proposed method to different distortions such as rotation, scale, and translation is an important direction that we will investigate in the future. Another direction is to analyse the complexity of the evolved descriptor and adopt a multi-objective approach to evolve a low-cost and high-performance descriptor. The effect of the features extracted by the proposed method on the performance of different classifiers will be investigated in the future.

7. REFERENCES

[1] W. Albukhanajer, J. Briffa, and Y. Jin. Evolutionary multiobjective image feature extraction in the presence of noise. *IEEE Transactions on Cybernetics*, pages 1–12, 2014.

[2] D. Androutsos, K. Plataniotis, and A. Venetsanopoulos. Distance measures for color

image retrieval. In *Proceedings of the International Conference on Image Processing*, volume 2, pages 770–774. IEEE, 1998.

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

[4] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[5] 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.

[6] 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.

[7] S. Hindmarsh, P. Andreae, and M. Zhang. Genetic programming for improving image descriptors generated using the scale-invariant feature transform. In *Proceedings of the 27th International Conference on Image and Vision Computing New Zealand*, pages 85–90. ACM, 2012.

[8] G. Holmes, B. Pfahringer, R. Kirkby, E. Frank, and M. Hall. Multiclass alternating decision trees. In *Proceedings of the 13th European Conference on Machine Learning*, pages 161–172. Springer, 2002.

[9] S. Keerthi and C.-J. Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation*, 15(7):1667–1689, 2003.

[10] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[11] 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.

[12] K.-L. Lim and H. Galoogahi. Shape classification using local and global features. In *Proceedings of the 4th Pacific-Rim Symposium on Image and Video Technology*, pages 115–120. IEEE Computer Society, 2010.

[13] 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, 2001.

[14] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, pages 1150–1157. IEEE, 1999.

[15] S. Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013.

[16] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[17] T. Ojala, M. Pietikäinen, and D. Harwood. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In *Proceedings of the 12th International Conference on Pattern Recognition*, volume 1, pages 582–585. IEEE, 1994.

[18] G. Olague and L. Trujillo. Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. *Image and Vision Computing*, 29(7):484–498, 2011.

[19] C. B. Perez and G. Olague. Evolutionary learning of local descriptor operators for object recognition. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 1051–1058. ACM, 2009.

[20] W. Smart and M. Zhang. Using genetic programming for multiclass classification by simultaneously solving component binary classification problems. In *Proceedings of the 8th European Conference on Genetic Programming*, pages 227–239. Springer, 2005.

[21] A. Song, T. Loveard, and V. Ciesielski. Towards genetic programming for texture classification. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence*, volume 2256 of *Lecture Notes in Computer Science*, pages 461–472. Springer, 2001.

[22] W. A. Tackett. Genetic programming for feature discovery and image discrimination. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 303–311, 1993.

[23] S. Trenn. Multilayer perceptrons: Approximation order and necessary number of hidden units. *IEEE Transactions on Neural Networks*, 19(5):836–844, 2008.

[24] L. Trujillo and G. Olague. Automated design of image operators that detect interest points. *Evolutionary Computation*, 16(4):483–507, 2008.

[25] M. Tuceryan and A. K. Jain. Texture analysis. In C. H. Chen, L. F. Pau, and P. S. P. Wang, editors, *Handbook of Pattern Recognition & Computer Vision*, pages 235–276. World Scientific, 1993.

[26] P. Whigham and G. Dick. Implicitly controlling bloat in genetic programming. *IEEE Transactions on Evolutionary Computation*, 14(2):173–190, 2010.

[27] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[28] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, 2005.

[29] L. Zhang, L. Jack, and A. Nandi. Extending genetic programming for multi-class 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.

[30] M. Zhang and V. Ciesielski. Genetic programming for multiple class object detection. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, volume 1747 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 1999.

[31] M. Zhang, V. 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.

[32] M. Zhang and M. Johnston. A variant program structure in tree-based genetic programming for multiclass object classification. In *Evolutionary Image Analysis and Signal Processing*, volume 213 of *Studies in Computational Intelligence*, pages 55–72. Springer, 2009.