In Cagnoni, Lutton and Olague [Eds]. *Genetic and Evolutionary Image Processing and Analysis* Hindawi Publishing Corporation, 2007.

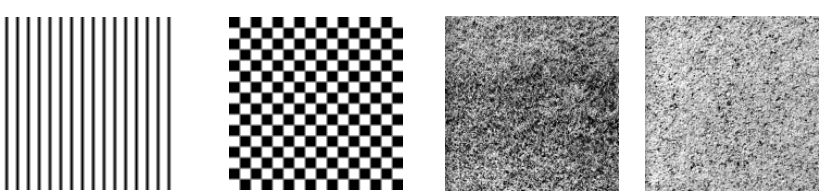
10 Visual Texture Classification and Segmentation by Genetic Programming

Vic Ciesielski, Andy Song and Brian Lam

10.1. Introduction

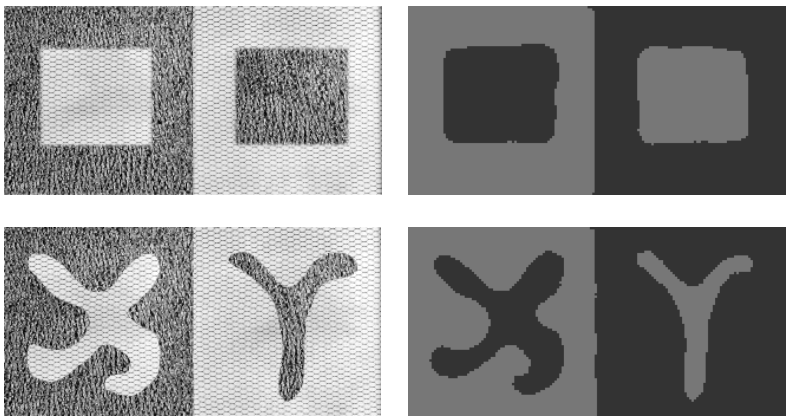
While there is a considerable history of work on visual texture, the definition of texture is still imprecise. However, it is generally agreed that a texture is spatially homogeneous and contains repeated visual patterns. In synthetic textures, such as horizontal lines, vertical lines or a checkerboard, the basic structure is repeated exactly. In natural textures, such as grass, wood, sand or rocks, there is some random variation in size, shape, intensity or colour in the repetitions of the basic structure. Figure 10.1 shows some examples of artificial (a,b) and natural (c,d) textures.

Texture information is potentially very useful in computer vision applications such as image/video retrieval, automated industrial inspection and robot navigation. However, currently deployed systems do not use texture, primarily because current algorithms result in unacceptably long computation times. Fast and accurate texture recognition could have a major impact on the design of future vision systems.



(a) Vertical Lines (b) Checkerboard (c) Grass (d) Sand

Figure 10.1. Examples of Synthetic and Natural Textures



(a) Original Image
Calf Leather and Netting

(b) Segmented Image

Figure 10.2. Examples of Segmentation by Texture

From the perspective of computer vision there are two main problems relating to texture: classification and segmentation. The classification task assumes we have images of single textures and we are required to distinguish them. For example if we have images like figures 10.1c and 10.1d, or sub images cut out from them, we need to identify which are grass and which are sand. In a texture segmentation task we have images containing several textures and are required to identify the regions in the image occupied by each texture, as shown in figure 10.2. The input image contains arbitrary regions of two different textures and the segmented image is a two colour image with one colour used for each texture. A problem in which we are required to retrieve all images containing, for example, a sandy beach, is a variant of the segmentation problem in which only one texture is of interest.

Texture classification and segmentation problems can be supervised, where the set of textures is known in advance, or unsupervised, where it is not known. This paper is concerned with supervised situations.

10.1.1. Conventional Approach to Texture Classification

The conventional approach to texture classification is shown in figure 10.3. The task is to assign a texture label to an image like one of those in figure 10.1. A two step procedure is used. In the first step a vector of features, based on human derived theories and models of texture, is computed. In the second step a classifier such as a decision tree, neural network or nearest neighbour classifier is used to assign a class to the feature vector (A large number of classifiers and their implementations are described in [1]). Most of the research in texture classification is focused on the first stage and

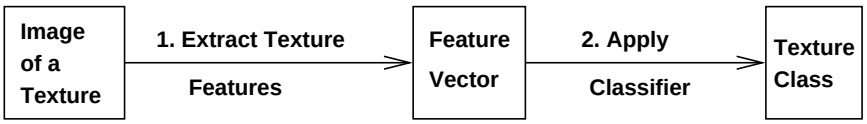


Figure 10.3. The Conventional Approach to Texture Classification

a large number of different ways of getting useful, highly discriminatory features have been investigated. These include Haralick features [2], Laws masks [3] and various wavelet transforms [4, 5, 6, 7]. The most widely used features over the last 35 years are the Haralick features which are based on an intermediate data structure called the grey level co-occurrence matrix. Over the years there have been many refinements to the basic Haralick approach, for example [8]. More recently, features based on wavelets have generated considerable interest. Most new algorithms for texture features are compared against the Haralick features. Work on texture up until 1993 is reviewed in [9], the most recent review of texture analysis. In some very recent work there has been a focus on developing models of a texture unit or "texton" and using the models to develop classifiers [10, 11]. However expensive computation is required as various Gabor and Laplacian transforms at a number of locations, scales and orientations are used.

The conventional approach has three main drawbacks. Firstly, there is no universal set of optimal texture features. Some of the features work very well for some textures and very badly for others. While there are many texture features, it is not clear which combination of features will suit a given problem and a trial and error process is needed for each new texture classification/segmentation task. Secondly, some of the approaches generate an enormous number of features, perhaps more than there are pixels in the image. This necessitates complex dimensionality reduction in feature space. Thirdly, most of the texture feature extraction algorithms are computationally expensive. They require the generation of Fourier-type transforms or other complex intermediate data structures and then additional computation on these structures. In this paper we show that the use of the genetic programming techniques can overcome some of these drawbacks.

10.1.2. Aim

The aim of this paper is to describe a number of ways in which genetic programming can be used in texture classification and to show how some of the classifiers can be used for fast, accurate texture segmentation.

There are at least three ways in which genetic programming can be used in texture classification. They are

(1) Use conventional feature extraction followed by evolving a genetic programming classifier, that is, use a genetic program as a classifier in step 2 of figure 10.3.
(2) Use a one step procedure in which the classifiers are evolved directly from example images of the textures of interest, that is, replace steps 1 and 2 with a single step which does the classification directly from the pixels without any feature extraction.
(3) Evolve the actual feature extraction programs which can then be used with a conventional classifier or a GP classifier, that is, replace the human constructed feature extraction programs in step one with evolved feature extraction programs.

In sections 10.3 to 10.5 we describe these approaches. In section 10.6 we show how the fastest classifiers can be used for segmentation.

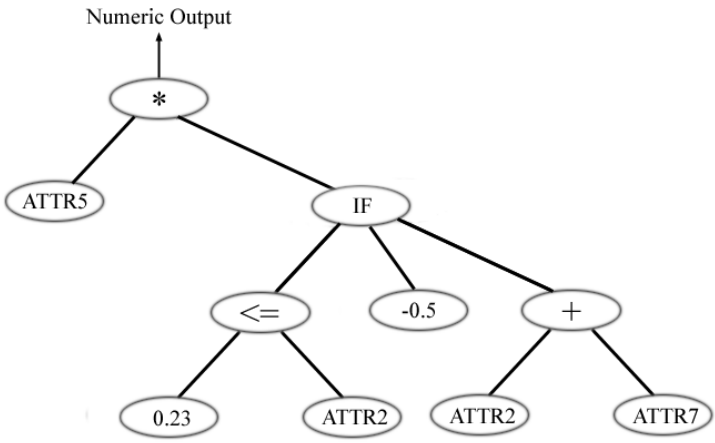
All of the texture images used in this paper are taken from the Brodatz album [12]. The original photographs were taken by a commercial photographer, Phil Brodatz, and intended for creative designers. The photographs have been digitised and have become a kind of de facto standard in texture research. We have used the images from [13]. Altogether there are 111 different textures labeled D1 to D112, with D14 missing.

10.2. Genetic Programming

Genetic programming is a methodology for obtaining computer programs to solve a particular problem by a process of simulated evolution. An initial population of programs is constructed. Each program is executed on the problem at hand and its success on the task, its fitness, is measured. A new population of programs is then constructed by selecting the fitter programs as parents and generating children by recombining selected parts of the parents (crossover) and/or making random changes to the parents (mutation). This process continues until the problem is solved or until some preset number of generations has been completed. If the process is working well, the programs will gradually become fitter and fitter through the generations until the problem is solved.

At the current state of the art the evolved programs are not in conventional programming languages such as C or Java. Rather, they are in languages which have been designed with restricted syntax and semantics so that two arbitrarily combined program fragments will form a syntactically valid executable program. Approaches include tree based genetic programming in which the programs are in a subset of LISP, linear genetic programming in which the programs are in an assembly language and Cartesian genetic programming in which the evolved program is a network of specialised computing nodes. Our work uses tree based genetic programming.

In tree based genetic programming, programs are represented as tree structures. An example tree and the corresponding code are shown in figure



Tree Form

```
   (* ATTR5
      (IF (<= 0.23 ATTR2)
[then]    -0.5
[else]    (+ ATTR2 ATTR7)
      )
   )
```

Text Form

Figure 10.4. A Program in Tree Based Genetic Programming

10.4. The internal nodes are functions and the terminals are inputs to the program. The tree is evaluated in a bottom up fashion and the value of the root node is the output of the program.

In specifying the configuration of a genetic programming run it is necessary to give the functions, the terminals, a method of evaluating fitness and a number of parameters for the evolutionary parameters. These include the population size, the maximum number of generations to compute if a solution is not found, the elitism rate (the percentage of best individuals in the current generation copied without change to the next generation), the crossover rate (percentage of individuals in the new population that are created by crossover) mutation rate (the percentage of individuals in the new population created by mutation) and the maximum permitted tree depth.

While there has been previous work on texture recognition using evolutionary computation techniques, for example [14], genetic programming for image processing tasks [15, 16] and genetic programming for texture

synthesis [17] there is no significant prior work on genetic programming for texture recognition.

10.2.1. Genetic Programming Classifiers

There has been prior work on evolving genetic programming classifiers [18, 19]. For a problem in which there are only two classes, for example images of grass and sand, the most straightforward approach is to use positive values of a program like the one shown in figure 10.4 as texture 1, e.g grass, and negative outputs as texture 2, e.g. sand. To obtain the classifier the available data is split into a training and test set, following the machine learning methodology for learning from examples. To get the fitness of an evolved program, it is applied to each example in the training data and the number of classification errors is counted. The fewer the errors the fitter the program. Once a program achieves a fitness of zero the evolutionary run can be terminated.

We have used a refinement of this basic method which is described in [18, 20]. In this refinement, called dynamic range selection, the real line is split into a variable number of ranges, not just two (less than 0, greater than 0) as in the straightforward approach. The range boundaries are evolved along with the classification program. In figure 10.5, for example, if the output of the program is less than -250 then the example is class 2, if the out put of the program is between -131 and - 107 it is class 1. These classifiers are more accurate and are evolved in fewer generations than the ones from the straight forward approach [18]. Also, they can be easily extended to more than two classes.

10.3. Two-Step Texture Classification, GP Classifier

In this approach conventional texture feature extraction programs are used to generate a feature vector, but a genetic programming classifier will be used in in second step of the process described in figure 10.3.

In determining which features to use there are thousands of options. As indicated in section 10.1.1 there have been many papers on approaches to extracting visual texture features, some of which generate hundreds of features. In our experiments we have used a selection of Haralick [2] and Gabor wavelet features[4, 7].

We used thirteen Haralick feature functions. 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 computed at four angles (0°, 45°, 90° and 135°) for five distance values (1, 3, 5, 7, 9). At each distance, the average of each feature for the four angles is used as an additional feature. This gives a total of $13 \times 4 \times 5 = 260$ Haralick features.

Table 10.1. Functions Used in the Genetic Programming Runs

Name	Return Type	Argument Types	Description
$+$	Dbl	Dbl Dbl	Arithmetic Addition
$-$	Dbl	Dbl Dbl	Arithmetic Subtraction
$\times$	Dbl	Dbl Dbl	Arithmetic Multiplication
%	Dbl	Dbl Dbl	Protected Division
IF	Bool	Bool Dbl Dbl	If $arg1$ is true return $arg2$ else return $arg3$
$\leq$	Bool	Dbl Dbl	True if $arg1 \leq arg2$
$\geq$	Bool	Dbl Dbl	True if $arg1 \geq arg2$
$=$	Bool	Dbl Dbl	True if $arg1 = arg2$
Between	Bool	Dbl Dbl Dbl	True if the value of $arg1$ is between $arg2$ and $arg3$

For the Gabor features, we have used five fragment sizes from 2×2 to 32×32 and six orientations (0°, 30°, 60°, 90°, 120°, 150°). For each combination of scale and orientation, the mean and the standard deviation of the transform coefficients are computed. Thus there are $5 \times 6 \times 2 = 60$ Gabor features extracted from each image.

In total, the feature vector for each image contains $260 + 60 = 320$ elements.

10.3.1. Configuration of Genetic Programming

We have performed 110 experiments on the Brodatz textures to determine how well one texture can be distinguished from the other 110. The details of the experiments are given below. The images used have been randomly cut out from the original 640×640 Brodatz texture images from [13]. The experiments have been done using the RMIT-GP package [21].
Images: 1110 sub images from the chosen texture make up class 1 and 10 random sub images from each of the other 110 make up class2, giving a total of 2210 examples per experiment.
Image Size: 64×64 pixels.
Number of Classes: 2
Error Estimation: 10-fold cross validation.
Functions As shown in table 10.3.
Terminals: As shown in table 10.2. The values of each extracted feature, together with a random number generator, comprise the terminals.
Fitness: Classification error.

Table 10.2. Terminal Set

Name	Return Type	Description
Random(-1, 1)	Double	Random constant
Feature[x]	Double	Value of Feature x, $0 \leq x \leq 295$

Parameters: Population size, 200; max generations, 50; elitism rate, 0.10, crossover rate, 0.85; mutation rate, 0.05; maximum tree depth, 30.

10.3.2. Results

The results for the textures pictured in this chapter are given in table 10.3. The reasons for choosing these textures are given in Section 10.5. For each classifier 10 runs were performed and the performance of the best evolved classifier is shown in the two columns labeled FE+GP (Feature Extraction, Genetic Programming Classifier). The averages over all 110 experiments are shown as the last line. For comparison, the results of using the same features with a well known, commonly used classifier, the C4.5 decision tree classifier, are shown in the columns labeled FE+C4.5. The J48 implementation of the C4.5 algorithm in the Weka machine learning toolkit was used with the default parameters [22]. The complete results for all 111 textures can be found in [23].

Over the entire image set the C4.5 classifier is slightly more accurate. However, there was considerably more over training with C4.5 than with the genetic programming classifier. The major drawback of the GP approach is the training time, which is considerably longer than C4.5.

Interestingly, when the same experiment was repeated with a set of synthetic binary textures such as Figure 10.1a-b, the GP classifier achieved 100% accurate classification on all test data while the C4.5 classifier made a small number of errors.

10.4. One-Step Texture Classification

In this section we look at a novel approach to performing texture classification in one step. The texture images are given directly to the GP system and the classifiers are evolved from the image pixels directly, bypassing the feature extraction step.

10.4.1. Configuration of Genetic Programming

Number of images: As in section 10.3.1.
Image Size: 64×64 as in section 10.3.1 except that the images were scaled to 1/4 of their original size.
Error Estimation: 10 fold cross validation.
Functions: As shown in table 10.3.
Terminals: As shown in table 10.2, except that there are 256 terminals and the terminal "Attribute[x]" returns the intensity value of the pixel at position $((x-1)\ mod\ 16, (x-1)/16)$ of the input image.
Fitness: Classification error.
Parameters: Population size, 200; max generations, 50; elitism rate, 0.10, crossover rate, 0.85; mutation rate, 0.05; maximum tree depth, 30.

Table 10.3. Comparison of Classification Accuracy on Textures Pictured in this Chapter.

		FE+C4.5		FE+GP		One Step GP	
No	Texture	Train	Test	Train	Test	Train	Test
D2	Fieldstone	98.72	92.14	93.79	93.15	84.26	80.38
D3	Reptile skin	99.77	98.03	97.65	96.80	95.56	94.13
D4	Pressed cork	99.86	98.85	99.67	99.54	86.55	86.10
D5	Expanded mica	99.22	94.01	93.79	91.78	83.28	82.76
D9	Grass	99.26	94.74	93.40	89.49	82.04	81.44
D12	Bark	99.49	93.74	91.90	89.95	82.76	74.72
D15	Straw	99.81	97.89	98.82	98.63	90.53	90.93
D16	Herringbone	99.81	98.76	99.41	99.54	84.53	80.81
D19	Woollen Cloth	99.26	94.74	93.01	92.69	84.72	84.78
D21	French canvas	99.95	99.77	100.00	100.00	97.58	96.12
D24	Calf Leather	99.77	98.12	98.23	98.17	89.62	89.46
D29	Beach Sand	99.72	96.71	97.06	95.89	88.31	84.34
D34	Netting	100.00	99.90	100.00	100.00	99.28	99.55
D38	Water	99.72	97.57	98.36	97.26	96.60	95.84
D57	Straw matting	99.86	99.26	99.73	99.08	91.25	87.13
D68	Wood grain	99.72	98.81	99.73	99.08	95.69	94.44
D84	Raffia	99.81	97.53	98.75	98.63	93.79	94.42
D92	Pigskin	99.54	96.75	95.88	95.89	83.81	82.36
D94	Brick Wall	99.45	96.84	97.51	96.80	93.14	92.19
D112	Plastic Bubbles	99.49	95.38	96.54	92.23	88.44	87.17
	Average of 110		95.67		94.17		87.86

Scaling of the images to 16×16 results in 256 terminals in a genetic programming run. Unscaled images would require $64 \times 64 = 4096$ terminals. This is beyond the capacity of our genetic programming system. We have found empirically that around 500 terminals is the limit. This is due to the increase in the size of the search space because of the large number of possibilities whenever it is necessary to choose a terminal.

The classification results for this approach for a sample of the 111 textures are shown in the last column of table 10.3. In 18% of the experiments, or 20 cases, accuracies above 95% were achieved. Of these 20 cases, five were classified nearly perfectly with accuracy above 99%. None of the cases reached 100% test accuracy. In three cases early termination occurred, that is, the evolutionary process terminated with 100% classification accuracy on the training data before 50 generations was reached. The one step classifiers tended to be more accurate when the texture images are roughly homogeneous in terms of the structure of the texture or intensity distribution and when geometric regularity (for example, bricks) was present. The one step classifiers were least accurate when the images are extremely inhomogeneous and the sub images cut from different parts of the original image vary significantly.

It can be seen from table 10.3 that the one step classifiers are generally less accurate than classifiers based on extracted features. This is not unexpected as the feature extraction algorithms are based on well thought

out human models and theories. What is surprising is that the evolved classifiers which are based solely on image pixels are so accurate over such a large variety of natural textures. In addition, the evolved classifiers are very fast compared to those using feature extraction. They involve the evaluation of a relatively simple arithmetic expression based on the values of a relatively small number of pixels. This suggests that these classifiers could be useful for texture segmentation in algorithms that require a classifier to be executed at each pixel position. The decreased accuracy could perhaps be offset by the knowledge that neighbouring pixels are likely to be in the same texture region. We come back to this issue in section 10.6.

A typical evolved program is shown in figure 10.5. The programs varied in size, but no successful program contained more than 700 nodes. A negative aspect is that the programs are too complex comprehend and the underlying algorithm cannot be recovered. It is not possible to tell whether some real texture regularities have been captured or whether some artifact of the training data has been captured. In order determine whether any texture regularities are being captured in the evolved programs we have carried with a number of experiments with simple binary textures involving horizontal, vertical and diagonal lines. We have used a small number of terminals and functions and encouraged the evolution of small programs by using a size penalty in the fitness function. Analysis of the evolved programs revealed that the evolved programs could be interpreted as texture masks which did capture regular differences between the textures. This work is described in [23, 24]. This result, together with the large number of training images used with the Brodatz textures, gives us confidence that texture regularities are being discovered even though the programs are too complex for analysis.

10.5. Two-Step Texture Classification, Evolved Features

In this section we return to the two step procedure. However, in this case we replace the hand crafted texture feature extraction programs by evolved programs. The feature extraction programs will be evolved from a selection of textures which we call the learning set. The evolved features will then be evaluated using a train-and-test methodology on the learning set and on a problem involving a different set of Brodatz textures.

Following the successful use of direct pixel inputs in the one step approach, our original intention was to use pixel inputs from the examples of the learning set for the evolution of the feature extraction programs. However, feature extraction programs evolved in this fashion were not very discriminatory. From an analysis of the evolved programs and results it appeared that some kind of aggregation of pixel information was needed. We subsequently experimented with histograms, which give a very coarse aggregation, and discovered that this worked surprisingly well. The methodology and results are described in this section.

```
(* -0.208533 (if (n= (- ATTR39 ATTR69) (+ ATTR114 0.299995)) (- ATTR222
0.253174) ( + ( + (if (Between ( * ATTR189 ( + ATTR205 ATTR74)) ATTR164
ATTR12) ATTR108 ( if (Between ( - (* 0.253174 ATTR129) 0.9875) 0.128228
(/ ATTR189 0.940897)) ( + ATTR39 (if (Between -0.947054 ATTR164 ATTR12)
ATTR160 ATTR124)) (* 0.5979 0.230855))) ( + ( + (if (Between (+ ATTR205
ATTR74) ATTR164 ATTR12) ATTR108 ( if ( Between -0.947054 ( if ( Between
-0.947054  ATTR164  ATTR12) ATTR160  ATTR124)  ATTR12) ( + ATTR39 (if (
Between -0.947054  ATTR164 ( - (/ ATTR149 ( if (Between -0.947054 (if (
Between -0.947054  ATTR164  ATTR12) ATTR160  ATTR124)  ATTR12)  ATTR108
0.135714)) (* ATTR164 ATTR129))) ATTR160 ATTR124))(* 0.5979 0.230855)))
(if (Between (* ATTR189 (+ (if (Between (- (* 0.253174 ATTR129) 0.9875)
0.128228 ( / ATTR189 ( - 0.987674 0.747389))) ( + ATTR39 ( if ( Between
-0.947054  ATTR164  ATTR12)  ATTR160  ATTR124)) ( *  0.5979  0.230855))
ATTR74)) ATTR164 ATTR12) ATTR205 ( + ( + ( if (Between (* (* ATTR189 (+
ATTR205 ATTR74)) ( + ATTR205  ATTR74 )) ATTR164  ATTR12 ) ATTR108 (if (
Between (- (* 0.253174 ATTR129) 0.9875) 0.128228 (/ ATTR189 (- 0.987674
0.747389))) ( +  ATTR39 (if ( Between -0.947054 ATTR164 ATTR12) ATTR160
ATTR124)) ( * 0.5979 0.230855))) ( + ( + ATTR160 (if (Between -0.947054
ATTR164 ATTR12) ATTR160  ATTR114))  ( * 0.253174  ATTR129))) ATTR205)))
(* 0.253174 ATTR129))) ATTR205)))

Ranges:
Class 1: D112          : -131 ~ -107 and  24 ~136 and  175 ~ +250
Class 2: Other Textures: -250 ~ -132 and -106~ 23 and  137 ~ 184
```

Figure 10.5. A Typical Evolved Program. ATTRx is the value of the pixel at position $((x-1)\ mod\ 16, (x-1)/16)$ of the input image.

The histogram of an 8 bit grey level image is a vector of length 256 in which element i gives the number of pixels which have a grey level value of i. In a histogram all information about the spatial arrangement of pixels is lost, which suggests that histograms might not be very useful for texture.

To evolve the feature extraction programs we have selected a learning set of 13 textures, shown in figure 10.11a-m. These textures were chosen to enable comparison with [25] in which 18 texture feature extraction algorithms are compared on a set of classification problems using this data set.

We have evolved 78 feature extraction programs, one from each pair of textures shown in figure 10.11. The evolutionary procedure described in the next section was repeated 78 times, once for each pair of combinations of the 13 textures in the learning set.

10.5.1. Configuration of Genetic Programming

Images: Two textures selected from the 13 shown in figure 10.11.
Image Size: 64×64 cut randomly from the original large images.
Number of images: 80 images of texture 1 and 80 of texture 2.

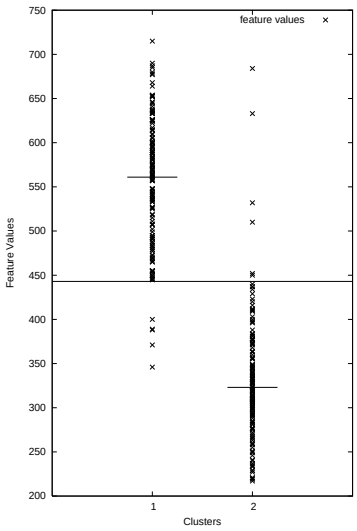


Figure 10.6. Feature Space for Two Texture Classes, D9 and D12

Functions: One only $\{+\}$. We began with the function set used in the earlier experiments (table 10.3). However, we found that using fewer operators resulted in programs that were just as accurate, but which were easier to comprehend.
Terminals: Histogram$[i], 0 \leq i \leq 255$.
Fitness Evaluation: A feature extraction algorithm is considered useful if the feature values result in high classification accuracy. This will occur if the feature values computed for each class are well separated in feature space. Thus, to evolve feature extraction algorithms, we need a way to implement the intuition that "the better the separation, the better the fitness". We have done this by computing the overlap between clusters generated by the K-means clustering algorithm. An example of this, for the case where there are two texture classes in the learning set is shown in figure 10.6. To get the data shown in the figure a program in the population has been evaluated on a learning set of 160 images which consist of 80 examples of texture (a) from figure 10.11 and 80 examples from texture (b). The averages of the feature values for each class give cluster centroids at 561 and 323. The mid point of the two centroids is the cluster boundary, that is 443. There are 4 cluster1 feature values below the boundary and 6 cluster2 features above it, thus 10 points are incorrectly clustered. Equivalently, it can be considered that there are 10 errors. This represents quite good discrimination. In contrast, if there were 150 errors, the discrimination would be very poor.
Parameters: Population size, 100; max generations, 200; elitism rate, 0.02, crossover rate, 0.78; mutation rate, 0.28; maximum tree depth, 9.

10.5.2. Results

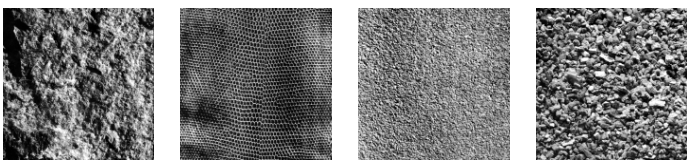
The 78 feature extraction programs were used on a 4 class problem involving the textures shown in figure 10.7 to generate a feature vector of length 78. Note that these are not in the learning set. There were 33 training images and 67 test images. Using a nearest neighbour classifier, the evolved features gave a test accuracy of 100% while the Haralick features gave a test accuracy of 95.5%. This result suggests that some generalisation has occurred, the evolved programs have captured some texture regularities that apply not just to the learning set but to other textures as well.

We have tested our evolved features on the two benchmark texture classification problems used in [25]. In [25], 18 human derived texture feature extraction algorithms are compared on two problems. The first problem is a 13 class problem involving the textures shown in figure 10.11a-m. On this problem the evolved features outperformed 5 of the human derived methods. The second problem is a 15 class problem involving textures from a different texture data base, the Vistex data base. Vistex problems are generally considered to be more difficult than Brodatz problems. This is because textures in a class are at different resolutions. In the Brodatz textures, images with three different sizes of bricks would be three different texture classes. In the Vistex textures they are one class. On this problem the evolved features outperformed 8 of the human derived methods. Full details of the results are in [26].

Using only histograms means that spatial relationships between pixels are being ignored. Two quite different textures could give similar histograms. Ways of incorporating spatial relationships still needs to be investigated.

10.5.3. Analysis of Evolved Feature Extraction Programs

Since + is the only function, all of the evolved algorithms are sums of the number of pixels at certain grey levels. For example, using grass (D9) as class1 and bark (D12) as class2 gives the feature extraction program X109 + 2*X116 + 2*X117 + X126 + 2*X132+ X133 + 2*143 + X151 + X206 + X238 + 3*X242 + X254, where X*nnn* represents the value of the histogram at grey level *nnn*. If we examine the histograms of two images, shown in



D2 D3 D4 D5

Figure 10.7. Four Brodatz Textures Used for Testing Evolved Features

figure 10.8, we can see the program has made use of the points between grey level 100 and grey level 150 and those above grey level 200 where class1 is significantly different from class2.

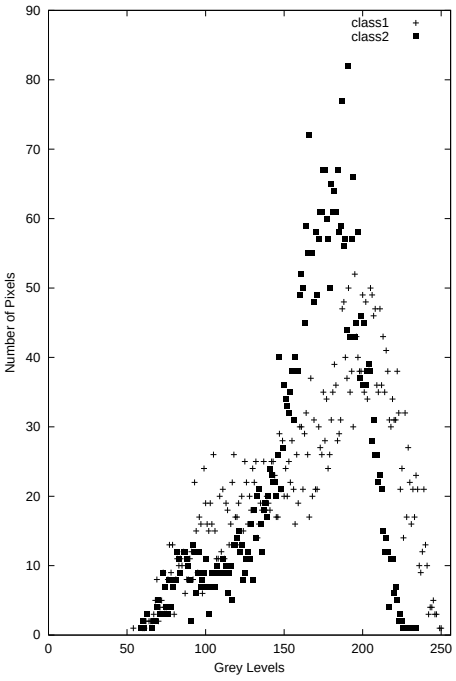
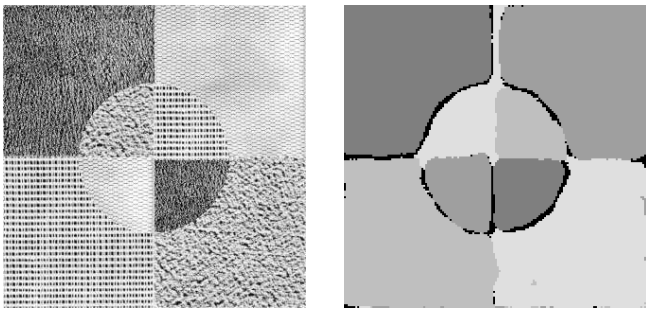


Figure 10.8. Histograms of D9 (Class1) and D12 (Class2) Textures

10.6. Texture Segmentation

The one step texture classifiers described in section 10.4 can be readily adapted for texture segmentation. If it is known that there are only two textures in the image to be segmented, as in figure 10.2 for example, then a binary classifier can be evolved from small windows cut out from each of the textures. In the segmentation step the classifier is then evaluated at each pixel position in the image to be segmented and each pixel given a colour based on the output of the classifier. This is what has been done to achieve the segmentation shown in figure 10.2. A window size of 16 was used.

The choice of window size is important, it cannot be too small or too big. Since a texture T is composed of repeated visual patterns, sub image should also exhibit the texture. If we take smaller and smaller cutouts these should still be texture T. However, the cutout cannot be too small, if the



Original Image, size 320 × 320 Segmentation Output

Figure 10.9. Segmentation of a Mosaic of Four Textures, D21, D24, D34, D57

D24 vs Others D34 vs Others D21 vs Others D57 vs Others

Figure 10.10. Areas Claimed by Each Classifier: (Light:"My-texture", Dark: "Not-my-texture)

cutout is smaller then the basic repetitive unit we no longer have texture T. For example, if we cut figure 10.1d into quarters then we still have a recognisable sand texture. If we cut those quarters into quarters we still have sand. However, if we cut too many times, a sub image will no longer look like sand. As another example consider bricks. As long as the cutout is at least as big as a brick we still have a brick texture, if it is smaller, the regular rectangular pattern of a brick texture is lost. In evolving one step classifiers for a segmentation task it is important that the window size is not smaller than the size of the basic repeating unit. If the window size is too big, there will be many ambiguous pixel labels around the boundaries between textures. For natural textures such as those in the Brodatz album it is difficult to determine an optimal window size apriori. We have empirically fixed on 16.

If there are more than two textures in the image to be segmented, as in the left hand image of figure 10.9 where there are four, the segmentation algorithm is more complex. We need 4 "me-vs-everybody-else" classifiers of the kind described in sections 10.3 and 10.4. Each of the classifiers is evaluated at each pixel position, with the pixel of interest at the centre of the window. Each classifier labels each pixel as "my-texture" or "not-my-texture". The final label or colour is then determined as follows: If a pixel

has 4 "not-my-texture" labels it is rendered in black. If it has 3 "not-my-picture" labels and 1 "my-texture" label then it is labeled and rendered as the claiming texture. If it is claimed by 2 or more textures it receives the label of the classifier that achieved the highest accuracy on the training data. The output of this algorithm is shown in the right hand image of figure 10.9. In figure 10.10 the areas claimed by each of the classifiers are shown in grey and the areas not claimed are shown in black. It can be seen that all but the last are quite accurate. Even though the last classifier has made some mistakes, the final segmentation is still very accurate because of the voting procedure.

10.6.1. Segmentation Speed

The classifiers generated by the single-step approach are relatively small. No classifier for Brodatz textures was observed to have more than 700 functions, which are simple operators like $+, -, *, /$ and IF. This characteristic enables a fast voting strategy which can lead to much faster segmentation than the conventional "feature extraction plus classification" approaches. In contrast, feature extraction algorithms are much more complex. Using Haralick features [27] as an example, at least one 256×256 co-occurrence matrix needs to be generated in computing the features for an image with 256 gray levels. One of the simplest Haralick features, energy, is computed as

$$\sum_{i=0}^{255} \sum_{j=0}^{255} M^2(i,j)$$

where $M(i,j)$ denotes one cell of the matrix. Such a calculation requires 256×256 repetitions of "+" and "*". Therefore more than $100,000$ operations are needed. Thus extracting Haralick features is much more expensive than executing GP generated classifiers. Even if the construction of the co-occurrence matrix and the cost of the subsequent classification step of the conventional approach required zero computation, the single-step classifier would still be much faster than the two-step approach.

In a texture segmentation task the feature extraction and classification must be performed many times. Consider a situation where the binary segmentation algorithm described above is being used. We have measured the time for performing one calculation of the Haralick features on a Sun SPARC computer and found that it is approximately 0.7 seconds. This means that the time to perform a segmentation of 256×256 gray level image using a window size of 16×16 is approximately

$$256^2 \times 0.7 \approx 45,000(seconds)$$

There are various ways of speeding up the approach, some heuristics are described in [28]. One method is to evaluate only every 2nd or 3rd pixel

position. Another is to use a split-and-merge approach where the image is recursively split into quarters to some predetermined size and the texture label of each fragment computed. Adjacent areas of the same texture are then merged into one region. If the split is limited to only 3 levels features are required to be computed on four 128×128 quarters for the first split, on sixteen 64×64 smaller quarters for the second split and on sixty-four 32×32 squares for the third split. The total CPU time for computing these features can be estimated as

$$4 \times 4.58 + 16 \times 4.10 + 64 \times 1.70 = 192.72(seconds)$$

The times for computing Haralick features for images of size 128×128, 64×64 and 32×32 on the SPARC computer are 4.58, 4.10 and 1.70 seconds respectively.

Clearly, whether the voting strategy or the split-and-merge strategy is used, the times taken on the SPARC computer for computing these features are much longer than 2.58 seconds, the time for segmenting a 320×320 image by our proposed algorithm. Furthermore, it is not hard to see that with only three levels of split, where 32×32 is the smallest size, the output will be too coarse to produce smooth boundaries. Further splitting requires even more computation time. To achieve an equivalent segmentation performance, the time spent by a conventional method based on Haralick features or Gabor features would be more than one hundred times the time spent by our method. There are many reports in the literature which describe long computation times for texture segmentation. Jain and Kalle reported 122 seconds of processing time on a SUN Sparc-10 workstation to segment a 256×256 gray-level image using Gabor filters and 109 seconds processing time by their proposed method [29]. Chang et al. evaluated different texture segmentation algorithms and reported the run times of "feature extraction plus classification" on a Sun Ultra Sparc. All of the reported times were more than 28 minutes [30]. In 2002, Chen and Chen proposed a new feature for fast texture segmentation [31]. The runtime for their experiment of segmenting 256×256 images on a Pentium III-500 PC was about 1 minute. Although the run times reported in the literature are measured under different conditions and not suitable for direct comparison, it is clear that the proposed method requires much less computation time since there is no computational expensive feature extraction.

10.7. Conclusions

We have described 3 different ways in which genetic programming can be used in texture classification: (1) as the classifier in the conventional "feature extraction + classification" approach, (2) as a one step classifier which bypasses feature extraction and uses pixel values directly, and (3) as a method for evolving texture feature extraction programs to be used in the

classical two step procedure. Overall the genetic classifiers are competitive with conventional classifiers in terms of accuracy. The major drawback is that they require long computation times to get a classifier. However, once evolved, the classifiers are very fast. This result suggests that the genetic classifiers could be used in real time situations where speed is more important than accuracy. A drawback of the genetic programming classifiers is that the programs are hard to comprehend.

An exciting outcome of this work is the accuracy and speed of the one step classifiers. This offers the prospect of fast real time segmentation in application areas such as robot vision, industrial inspection and image/video retrieval, applications in which texture is currently not being used because the computational cost is too high.

Bibliography

[1] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco, California, Morgan Kaufmann, 2000.

[2] Robert M. Haralick. Statistical and structural approaches to texture. *Proceedings of the IEEE*, 67(5):786–804, May 1979.

[3] K. Laws. Rapid texture identification. In *Proceedings of SPIE Conference on Image Processing for Missile Guidance*, volume 238, pages 376–380, 1980.

[4] S. Livens, P. Scheunders, G. van de Wouwer, and D. Van Dyck. Wavelets for texture analysis, an overview. In *Sixth International Conference on Image Processing and Its Applications*, volume 2, pages 581–585, 1997.

[5] S. Arivazhagan and L. Ganesan. Texture classification using wavelet transform. *Pattern Recognition Letttters*, 24(9-10):1513–1521, 2003.

[6] Inna Stainvas and David Lowe. A generative probabilistic oriented wavelet model for texture segmentation. *Neural Processing Letters*, 17(3):217–238, 2003.

[7] S.E. Grigorescu, N. Petkov, and P. Kruizinga. Comparison of texture features based on Gabor filters. *IEEE Transactions on Image Processing*, 11(10):1160–1167, October 2002.

[8] Christoph Palm. Color texture classification by integrative co-occurrence matrices. *Pattern Recognition*, 37(5):965–976, May 2004.

[9] Mihran Tuceryan and Anil K. Jain. Texture analysis. In C. H. Chen, L. F. Pau, and P. S. P. Wang, editors, *Handbook of Pattern Recognition and Computer Vision*, chapter 2, pages 235–276. World Scientific, Singapore, 1993.

[10] Oana G. Cula and Kristin J. Dana. 3D texture recognition using bidirectional feature histograms. *Internationa Journal of Computer Vision*, 59(1):33–60, 2004.

[11] Song-Chun Zhu, Cheng-En Guo, Yizhou Wang, and Zijian Xu. What are textons? *International Journal of Computer Vision*, 62(1-2):121–143, 2005.

[12] Phil Brodatz. *Textures: A Photographic Album for Artists and Designers*. Dover, NY, 1966.

[13] Brodatz texture database. http://www.ux.his.no/˜tranden/brodatz.html.

[14] Tamas Sziranyi and Marton Csapodi. Texture classification and segmentation by cellular neural networks using genetic learning. *Computer Vision and Image Understanding*, 71(3):255–270, 1998.

[15] Riccardo Poli. Genetic programming for feature detection and image segmentation. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 110–125. Springer-Verlag, 1996.

[16] Brian J. Ross, Anthony G. Gualtieri, Frank Fueten, and Paul Budkewitsch. Hyperspectral image analysis using genetic programming. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1196–1203. Morgan Kaufmann Publishers Inc., 2002.

[17] Adam Hewgill and Brian J. Ross. Procedural 3D texture synthesis using genetic programming. *Computers and Graphics*, 28(4):569–584, August 2004.

[18] Thomas Loveard and Vic Ciesielski. Representing classification problems in genetic programming. In Jong-Hwan Kim, editor, *Proceedings of the 2001 Congress on Evolutionary Computation*, pages 1070–1077, Seoul, South Korea, May 2001. IEEE.

[19] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal. Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation*, 4(3):242–258, September 2000.

[20] Andy Song, Thomas Loveard, and Vic Ciesielski. Towards genetic programming for texture classification. In Markus Sumpter, Dan Corbett, and Mike Brooks, editors, *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence, LNAI 2256*, pages 461–472, Berlin, December 2001. Springer-Verlag.

[21] RMIT-GP: The RMIT University Genetic Programming System. School of Computer Science and Information Technology, http://www.cs.rmit.edu.au/~vc/rmitgp.

[22] Weka machine learning project. Department of Computer Science, The University of Waikoto http://www.cs.waikato.ac.nz/~ml/.

[23] Andy Song. *Texture Classification: A Genetic Programming Approach*. PhD Thesis, RMIT, Department of Computer Science, 2003. http://www.cs.rmit.edu.au/~asong/thesis.pdf.

[24] Andy Song and Vic Ciesielski. Texture classifiers generated by genetic programming. In Xin Yao, editor, *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 1, pages 243–248, Honolulu, May 2002. IEEE.

[25] P. Wagner. Texture analysis. In B. Jahne, H. Haussecker, and P. Geissler, editors, *Handbook of Computer Vision and Applications*, volume 2, chapter 12, pages 275–308. Academic Press, San Diego, 1999.

[26] Brian Lam and Vic Ciesielski. Discovery of human competitive image texture feature extraction programs using genetic programming. In Kalyanmoy Deb et al., editor, *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO2004)*, volume 2, pages 1114–1125. Springer, June 2004.

[27] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions On Systems, Man, and Cybernetics*, SMC-3(6):610–621, November 1973.

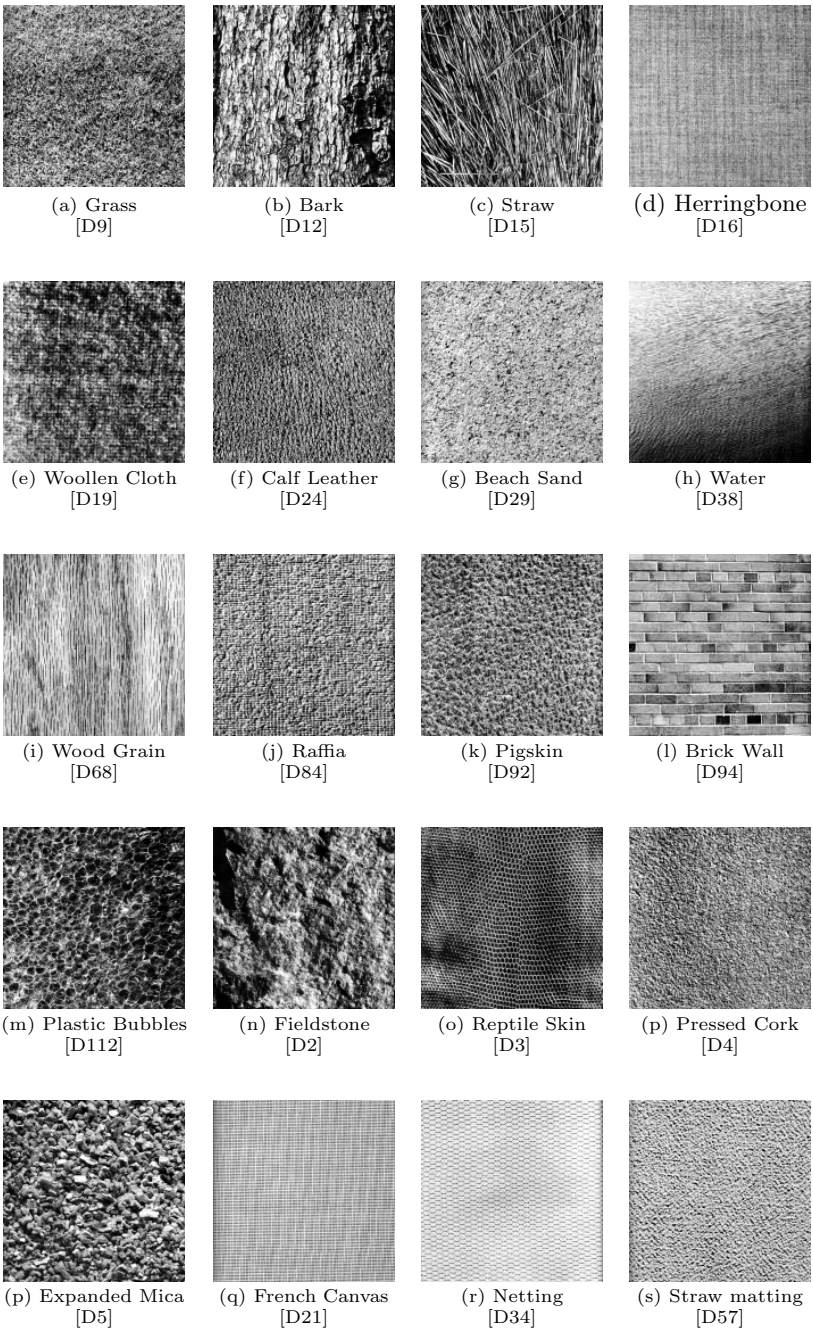
[28] Nikos Paragios and Rachid Deriche. Geodesic active regions and level set methods for supervised texture segmentation. *International Journal of Computer Vision*, 46(3):223–247, 2002.

[29] Anil K. Jain and Kalle Karu. Learning texture discrimination masks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:195–205, 1996.

[30] K.I. Chang, K.W. Bowyer, and M. Sivagurunath. Evaluation of texture segmentation algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1999*, volume 1, pages 294–299, 1999.

[31] Kan-Min Chen and Shu-Yuan Chen. Color texture segmentation using feature distributions. *Pattern Recognition Letters*, 23(7):755–771, May 2002.

School of Computer Science and Information Technology, RMIT University, GPO Box 2476V, Melbourne, Vic 3001, Australia.
{vc,asong,blam}@cs.rmit.edu.au



(a) Grass [D9] (b) Bark [D12] (c) Straw [D15] (d) Herringbone [D16]

(e) Woollen Cloth [D19] (f) Calf Leather [D24] (g) Beach Sand [D29] (h) Water [D38]

(i) Wood Grain [D68] (j) Raffia [D84] (k) Pigskin [D92] (l) Brick Wall [D94]

(m) Plastic Bubbles [D112] (n) Fieldstone [D2] (o) Reptile Skin [D3] (p) Pressed Cork [D4]

(p) Expanded Mica [D5] (q) French Canvas [D21] (r) Netting [D34] (s) Straw matting [D57]

Figure 10.11. The Brodatz Textures Used in this Paper