A Comparison of Genetic Programming Feature Extraction Languages for Image Classification

Mehran Maghoumi
Department of Computer Science
Brock University
St. Catharines, ON, Canada L2S 3A1
Email: mehran@maghoumi.com

Brian J. Ross
Department of Computer Science
Brock University
St. Catharines, ON, Canada L2S 3A1
Email: bross@brocku.com

***Abstract*—Visual pattern recognition and classification is a challenging computer vision problem. Genetic programming has been applied towards automatic visual pattern recognition. One of the main factors in evolving effective classifiers is the suitability of the GP language for defining expressions for feature extraction and classification. This research presents a comparative study of a variety of GP languages suitable for classification. Four different languages are examined, which use different selections of image processing operators. One of the languages does block classification, which means that an image is classified as a whole by examining many blocks of pixels within it. The other languages are pixel classifiers, which determine classification for a single pixel. Pixel classifiers are more common in the GP-vision literature. We tested the languages on different instances of Brodatz textures, as well as aerial and camera images. Our results show that the most effective languages are pixel-based ones with spatial operators. However, as is to be expected, the nature of the image will determine the effectiveness of the language used.**

I. Introduction

Pattern recognition and classification is a challenging computer vision problem. In basic terms, visual pattern classification involves the automatic identification of some image feature of interest. For example, one may want a computer program to automatically differentiate given different bitmap images of textures (wood grain, cloth, fur,...). The automatic recognition of more complex patterns is also possible, for example, fingerprints, faces, and targets in satellite images.

Genetic programming (GP) has been used in computer vision classification problems [1]. GP has been shown to be effective in a variety of image analysis tasks such as texture classification [2][3] and image and texture segmentation [3][4][5][6]. GP has also been applied to other complex computer vision problems, for example, and object detection [7][8][9], face detection [10], mineral identification [11], and medical image analysis [12]. More recently, GP has been applied to real-time computer vision problems in adaptive [13] and automated [14] object recognition.

Much of the aforementioned GP vision research use pixel-based classifiers. These classifiers involve GP expressions that examine features of a subject image centred around a pixel of interest, and determine a classification result for that pixel. To process an entire image, the GP tree is executed consecutively on all the pixels in an image. Although this has the potential of very precise pixel-level classification, it can be computationally expensive – and especially so for large GP expressions and high resolution images.

Song *et al.*'s work in GP vision uses a different approach – block (or region) classification [2][3][5][6]. In [2][3], the GP tree can access features of a (say) 32×32 square pixel region of an image. After one single execution of the classifier expression, a classification is made for that entire region of the image. After applying multiple classification decisions to different blocks of pixels in the image, an overall decision is ascertained for the whole image, in terms of a percentage of belief for the entire image. Using this technique, very good results were reported in [2][3]. Later work in [5][6] applied multiple classifier expressions to an image, and used a "voting strategy" to determine which classifiers identified each pixel most strongly during classifier expression applications. This permitted classification to be determined on a pixel basis. In all the above work, block classifiers are quite efficient to process, in that the classification expressions are composed of basic arithmetic operators – much more basic than expressions typically found in pixel-based languages.

The motivation of this paper is to more closely examine the effect of GP-based feature extraction languages on visual pattern recognition. In particular, we compare performance of block- and pixel-based classifiers on a different pattern recognition problems. We primarily consider Brodatz textures[15] as used in [2][3][5][6]. Brodatz textures are photographs of various natural and artificial patterns, for example, textile, stones, plant shapes, and others. We also consider a few satellite and camera images, in order to see how well the GP languages apply to alternative classes of images.

Although it is clear that block classifier languages are more computationally efficient than pixel languages, here we are primarily interested in comparing performance accuracy between the different pattern classification languages. Our curiosity is motivated by the fact that the block languages used in [2][3][5][6] are very simple compared to typical pixel-based languages found in the literature. These block languages use rudimentary mathematical operators, while pixel languages use a variety of statistical and image processing functions. We are motivated to see whether the higher-level operators in pixel languages contribute to more effective image classification.

The paper is organized as follows. The pattern classification problems to be examined are introduced in Section II. Four classification languages are defined in Section III, and other experimental details are given in Section IV. Results of our experiments are presented in Section V, and further discussed in Section VI. Section VII summarizes the paper, and discusses directions for future research.

978-1-4799-4503-0/14/$31.00 ©2014 IEEE

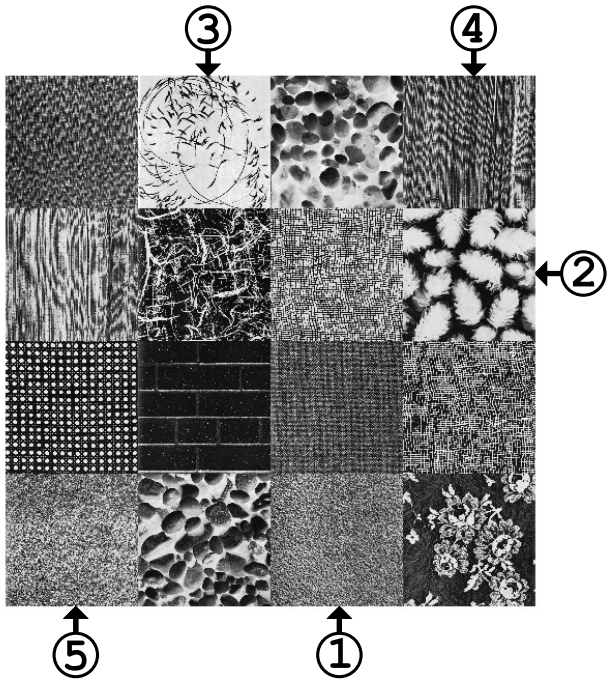


Fig. 1. Brodatz textures. Full image is 512×512 pixels, and each texture area is 128×128. Numbers indicate which textures are used for positive training cases.

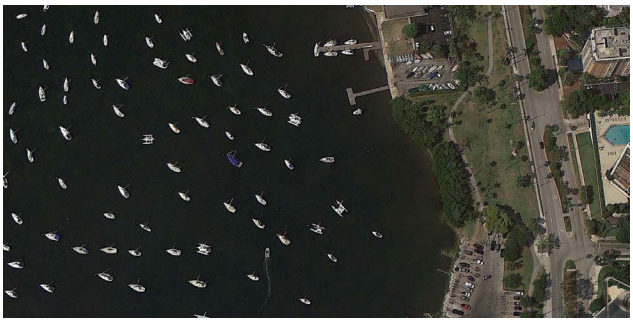


Fig. 2. Aerial image of boats. (1692×843 pixels.)

II. Pattern Classification Problems

The main vision problem of concern is the recognition of textures taken from the Brodatz texture data set[1][15], which were used previously in in [2][3][5][6]. Figure 1 shows the textures we have used. The 5 numbered textures are used separately as positive examples. For example, when texture 1 is a positive example, the remaining 15 textures are used as negative cases. The textures were selected to introduce various levels of difficulty. Some textures, such as 1 and 5, have very similar patterns while other textures, such as 3, are easily distinguished from others.

We also examine two other images, to give insights of the pattern recognition languages on other kinds of vision problems. We examine an aerial image of boats near a port obtained from the satellite view of Google Maps[2] (Figure 2). Our goal is to detect boats in the image. We also use a photo of a group of people (Figure 3), in which we want to identify

[1]http://www.ux.uis.no/∼tranden/brodatz.html
[2]http://maps.google.com/

Fig. 3. Group photo. (1280×720 pixels.)

human faces from the rest of the image. Ground truth images (see Fig. 5(a) and Fig 6(a)) were manually made to indicate the areas of the images with boats and faces – in other words, image areas to be considered as "positive" or true.

III. Classification Languages

A goal of this paper is to compare two different approaches to GP-evolved pattern classifiers – pixel classifiers and block (or region) classifiers. A pixel classifier makes a classification decision for a single pixel of interest, normally by examining spatial information in the surrounding image region of that pixel. A block classifier makes a decision for all the pixels within a region, based on information in that region. Pixel-based classifiers are more common in the GP literature (see Section I), while block-based classifiers are used by Song *et al.* [2][3][5][6].

In the following, we will describe both of these methodologies in detail. Since they are highly dependent upon the GP language used to define the classifier, the four GP languages studied (one block language, and three pixel languages) are also described.

A. Block classifier language

A block classifier language was presented by Song *et al.* [2][3], and has been used in other vision applications [5][6]. We implemented a block language resembling Song *et al.*'s original language. To perform block classification, multiple images are provided to the system: a target image for positive classification, the target for negative classification, and positive and negative test images. Training instances are created by randomly sampling 400 sub-image *blocks* each from the positive and negative target images. Our blocks are of size *32×32* pixels. The blocks contain various grey-scale (and possibly colour) information about the positive and negative target images, and comprise a feature vector for the GP system. Using the feature vector, an evolved GP tree composed of various mathematical and decision making functions can extract classification information from each block. This results in a binary classifier which, when applied to an image region, makes a classification decision for that entire region. The goal for evolution is to evolve a classifier that correctly classifies positive and negative image instances, according to the classification results for blocks in the image.

The block classification language is given in Table I, and uses functions and terminals standard in the literature. Att[x] represents the x^{th} pixel in the block (modulo total pixels in the block), and returns the RGB or grey-scale value. Strongly

TABLE I. BLOCK CLASSIFICATION LANGUAGE. (D=DOUBLE, B=BOOLEAN)

Name	Return type	Argument type	Description
Add	D	D	addition
Sub	D	D	subtraction
Mul	D	D	multiplication
Div	D	D	protected division
If	D	B, D, D	if a true then b else c
$\geq$	B	D, D	true if $a \geq b$
$\leq$	B	D, D	true if $a \leq b$
Between	B	D, D, D	true if $b < a < c$
Random	D	-	random constant, -1 $\leq c \leq$ 1
Att[x]	D	I	Value of attribute

typed GP is used for this and the other languages [16]. The language shown has Boolean and double data types. The root of each GP tree is double. Computed values greater than or equal to zero are interpreted as *true*, while negative are *false*.

When a block classifier makes a decision about a 32×32 region, all 1024 pixels in that region contribute to that classification. This represents a large region of the image, and would be too coarse for performing accurate pixel-based identification. (Recall that Song *et al.* [2][3] determine an overall classification for the whole image by inspecting multiple blocks of the image.) Therefore, for the majority of the pixels in an image, there are 1024 potential ways for assigning a classification to each pixel, depending on the placement of the 32×32 block overlaying that pixel. This poses some difficulty in comparing block classifiers with pixel classifiers, since the odds are good that there is at least a few block placements that give contradictory classifications for any given pixel. Therefore, in order to make a more meaningful comparison between block and pixel classifiers, we use the following approach. During testing, block classifiers are applied to images by placing the block area over every possible 32×32 region of an image (edges are not crossed, and so pixels close to edges have fewer block overlays). We then tally the number of times each pixel was identified as "true" within a tested block, scaled by the number of times that pixel was processed in total. This results in a percentage value that each pixel was classified as true. Should a pixel be classified as true the majority of time (i.e. threshold of 50%), then it is considered to be true with respect to performance measurements.

B. Pixel classifier languages

TABLE II. PIXEL LANGUAGE PART 1. (I=INTEGER, D=DOUBLE)

Name	Return type	Argument type	Description
Add	I/D	I/D	addition
Sub	I/D	I/D	subtraction
Mul	I/D	I/D	multiplication
Div	I/D	I/D	protected division
Neg	D	D	negation
Exp	D	D	e raised to the operand
IfGT	D	D,D,D,D	if $a > b$ then c else d
Max	D	D,D	maximum
Min	D	D,D	minimum
Sin	D	D	sine
Cos	D	D	cosine

Pixel-based classification works as follows. For a given data set, two images are provided to the GP system – an image to process, and a ground truth image. The ground truth image is marked to show the positive region(s) to be identified in the image. Using the ground truth, the system randomly samples 512 positive pixels and 1024 negative pixels from the input image. These *centre pixels* are then used for creating training instances. Using the coordinates of the centre pixels, a block of n×n pixels is formed around these pixels in a way that the centre pixels coincides with the coordinates ($\lfloor n/2 \rfloor$, $\lfloor n/2 \rfloor$). We refer to these blocks as *grids*. For each pixel in the grid, spatial filter values (average and standard deviation) are calculated and stored for later access by GP expressions. To speed up processing, we compute these values using NVIDIA CUDA [17].

TABLE IV. SUMMARY OF THE LANGUAGE VARIATIONS

Name	Spatial operations	Offsets	Block processing
Complete	×	×	
No Offset	×		
Raw Features		×	
Block Processing		×	×

During training, a binary classifier is evolved using positive and negative training instances. The raw pixel values, as well as the spatial filter values, form the feature matrix for the system. GP uses this matrix in conjunction with mathematical and decision making functions to evolve a classifier. Integer offsets can be used to extract features in the vicinity of the centre pixel. The decision made by the classifier is applied to the centre pixel. During testing, the classifier processes every pixel of the image. This contrasts to the block classifier, which assigns the classification to all the pixels in the region.

We define 3 pixel classification languages, which will use functions and terminals selected from Tables II and III. The languages use 3 data types: double, integer, and channel. Terminals (see Table III) include ephemeral random constants for integers and doubles, channel index, and random terminals (every access generates a new random value). Pixel values (RGB and/or grey-scale) are accessible, either directly for a centre pixel, or for a specified integer offset within the 32×32 block. The channel argument c specifies the particular channel (R, G, B, grey-scale) to retrieve. Grey-scale images have R, G, and B removed. Spatial data (average, standard deviation) can also be read for centre pixels and offsets near them, again for the specified channel. The area values (15, 17, 19) were determined by experimentation, as earlier attempts using smaller areas were not beneficial. Table II shows the remaining functions, which are standard in the literature. There are integer and double versions of the basic arithmetic operators.

The four languages used for the experiments are summarized in Table IV. *Spatial operations* refer to the average and standard deviation functions, while *offsets* include the colour and spatial operators that use (i,j) offsets. The first two languages (Complete, No Offset) thus use spatial operators, while the others do not. Raw Features is essentially the Block Processing language with extended mathematical operators, but to be used in a pixel-classification manner.

IV. EXPERIMENT SETUP

In all runs, the static range selection method [18] was used. A GP expression is evaluated for a pixel (block), and if the output is ≥ 0, it is considered a positive classification. This is compared to the ground truth image. The number of true positives and true negatives are then used to calculate the

TABLE III. PIXEL LANGUAGE PART 2. (I=INTEGER, D=DOUBLE, C=CHANNEL)

Name	Return type	Argument type	Description
c	C	-	channel index (0,1,2,3)
ERC	D	-	ephemeral random constant in the range [0, 1]
Random	I	-	random integer in the range [0, 31]
GridERC	I	-	ephemeral random constant in the range [0, 31]
Input colour	D	C	channel value of the selected pixel
$Avg_{k=15,17,19}$	D	C	average of $k \times k$ area
$Stdev_{k=15,17,19}$	D	C	standard deviation of $k \times k$ area
Input colour	D	C, I, I	channel value c at (i,j) offset
$GAvg_{k=15,17,19}$	D	C, I, I	average of $k \times k$ area of channel c, offset (i,j)
$GStdev_{k=15,17,19}$	D	C, I, I	standard deviation of $k \times k$ area of channel c, offset (i,j)

TABLE V. RUN PARAMETERS

Parameter	Value
Population size	1024
Generation size	50
Crossover rate	90%
Mutation rate	10%
Selection method	Tournament selection
Tournament size	4
Elites	2
Number of runs per experiment	20
Pixel block size	32×32
Training size, pixel langs.	512 +, 1024 -
Training size, block langs.	400 +, 400 -

fitness value of the individual in question:

$$fitness = \frac{TP + TN}{TOTAL} \times 100 \quad (1)$$

where TP is the number of true positives, TN is true negatives, and TOTAL is the total number of cases.

Table V summarizes the GP parameters used. Parameters were established via preliminary trials. Although we do not claim they are "optimal" in any sense, exploratory trial runs found that these parameters were effective for the GP languages and image data used. For example, we found that most runs tend to converge by generation 50. The size of positive and negative training sets reported in the table were found to be effective for the pixel and block languages. Even though the block language uses fewer training examples than the pixel languages, early trials showed that larger training sets tended to degrade block language performance. The training set size for the block languages is the same as in [2].

We use the Java-based ECJ genetic programming system as our GP engine [19].

V. RESULTS

Table VI shows the final test results of the experiments. The true positive and true negative scores are averaged over 20 runs, and do not include the training pixels. *Performance* is the average of the true positive and negative scores. We use this as a balanced measure of performance, because training is heavily biased towards negative pixels (there are more negative training examples than positive). *Best* is the performance score of the single top performing classifier found in the set of 20 runs. We have marked in boldface the best overall performing language within a statistical significance measure of 95% for each image studied. Statistical significance is measured with an unpaired t-test with unequal variance.

Although training scores are not reported here, the training and testing scores for true positives/negatives are closely matching in all experiments, and so over-training is unlikely to be occurring.

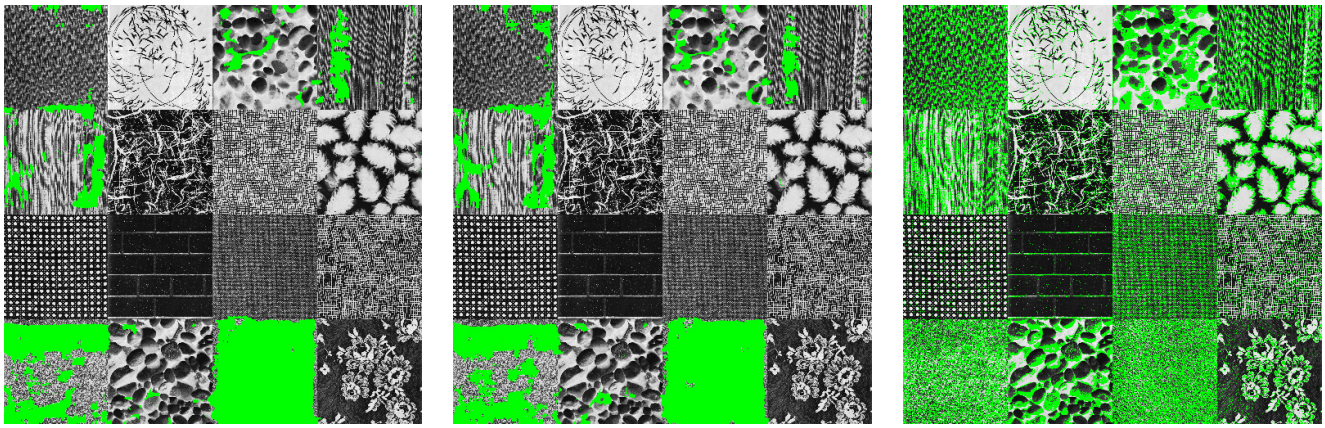
TABLE VI. EXPERIMENT RESULTS. ALL SCORES ARE %. "TEST +/-" SCORES ARE TESTING TRUE POSITIVE AND NEGATIVE, "PERFORMANCE" IS AVERAGE OF TEST + AND -, AND "BEST" IS THE SINGLE CLASSIFIER FROM THE 20 RUNS WITH THE TOP PERFORMANCE SCORE. ALL SCORES ARE AVERAGED OVER 20 RUNS (EXCEPT FOR "BEST"). TOP PERFORMING LANGUAGE SCORES (WITHIN STATISTICAL SIGNIFICANCE OF 95% VIA T-TEST) ARE HIGHLIGHTED IN BOLD.

		Language			
Image	**Test Scores**	**Complete**	**No Offset**	**Raw Feat.**	**Block Proc.**
Texture 1	test +	79.40	89.91	39.69	95.73
	test -	93.67	93.73	83.75	35.33
	performance	86.54	**91.82**	61.72	65.53
	best	94.14	93.48	65.55	80.43
Texture 2	test +	75.55	58.20	25.76	97.40
	test -	85.74	90.20	90.64	60.68
	performance	**80.65**	74.20	58.20	**79.04**
	best	89.77	82.64	62.18	84.12
Texture 3	test +	96.76	96.39	73.68	94.66
	test -	97.68	97.64	87.78	98.05
	performance	**97.22**	**97.01**	80.73	96.35
	best	98.19	98.24	81.42	96.95
Texture 4	test +	27.33	67.51	2.41	94.23
	test -	91.94	88.70	98.23	41.95
	performance	59.64	**78.11**	50.32	68.09
	best	83.94	86.05	51.62	69.17
Texture 5	test +	81.85	87.37	20.09	89.79
	test -	86.32	89.41	90.28	57.14
	performance	84.09	**88.39**	55.19	73.46
	best	91.05	90.70	62.69	79.27
Boats	test +	94.53	94.74	77.25	95.89
	test -	96.21	96.45	93.35	67.11
	performance	**95.37**	**95.59**	85.30	81.50
	best	96.29	96.81	88.04	92.68
Face	test +	90.94	95.28	87.90	94.97
	test -	92.65	94.26	89.86	64.69
	performance	91.79	**94.77**	88.88	79.83
	best	95.66	96.93	90.25	92.49
	Total wins	3	6	0	1

There is some variability apparent in performance for different languages and images. For example, for Texture 1, the Raw Features language is weaker at positive identification than the other languages. The Block Processing language, however, is weak in negative identification. Overall the spatial languages do better on Texture 1, with the No Offset language being the best performer.

Texture 4 was the most challenging image for the languages. The No Offset language was the best performer. Surprisingly, most solutions from the Complete language were weak, mostly due to poor positive recognition performance (low true positive scores).

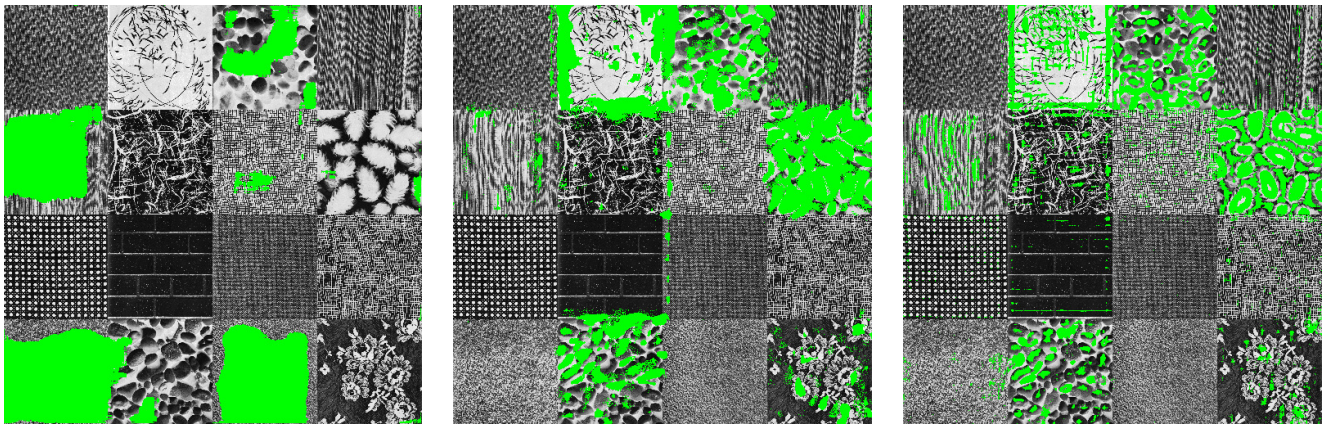
Summarizing the scores in Table VI, the No Offsets spatial language was the overall top performing language, having the best (statistically significant) average solution performance for 6 images. This is followed by the Complete spatial language (3 images), and the Block Processing language (1 image).



(a) Texture 1 - Complete
+ 94.7%, - 93.6%, tot 94.2%

(b) Texture 1 - No Offset
+ 94.4%, - 92.7%, tot 93.5%

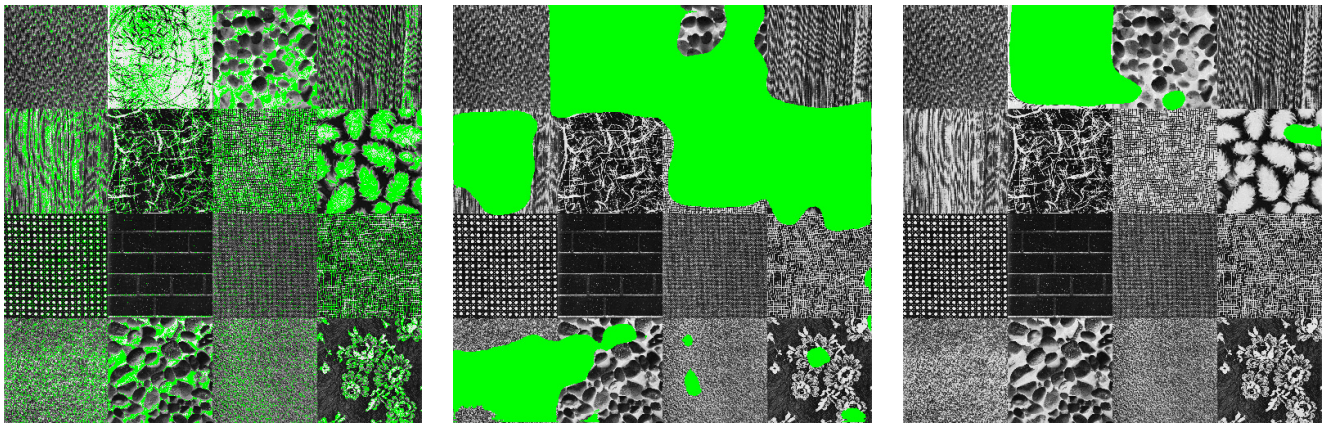
(c) Texture 1 - Raw Features
+ 51.6%, - 78.9%, tot 65.3%



(d) Texture 1 - Block Proc.
+ 74.0%, - 88.5%, tot 81.2%

(e) Texture 2 - Complete
+ 94.1%, - 85.5%, tot 89.8%

(f) Texture 2 - No Offset
+ 75.1%, - 90.3%, tot 82.7%

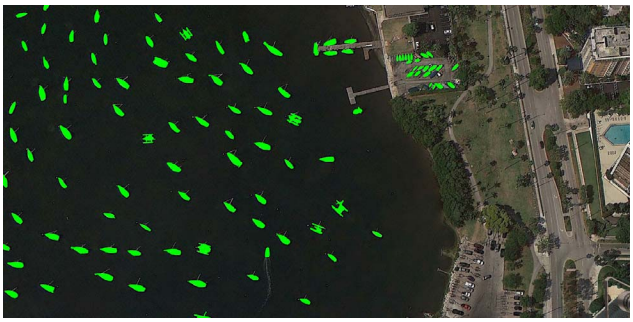


(g) Texture 2 - Raw Features
+ 41.2%, - 83.2%, tot 62.2%

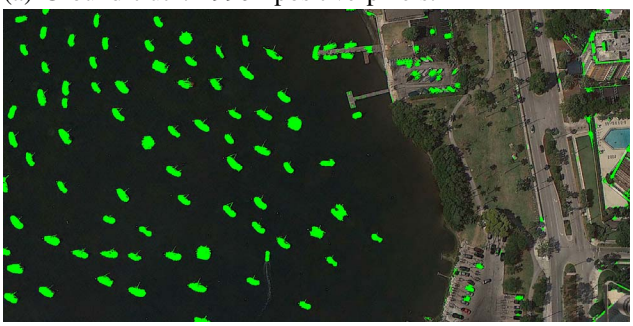
(h) Texture 2 - Block Proc.
+ 99.0%, - 70.8%, tot 84.9%

(i) Texture 3 - Block Proc.
+ 98.4%, - 97.6%, tot 98.0%

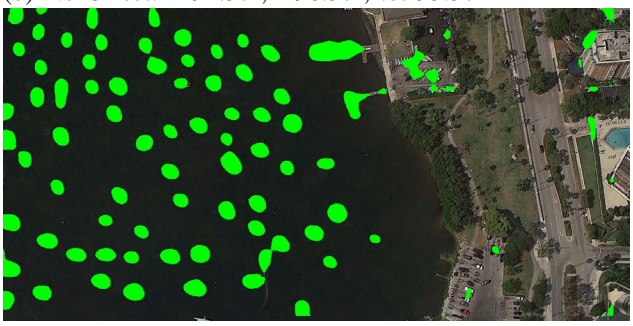
Fig. 4. Texture output images. See Fig. 1 for indication of textures 1, 2 and 3 used as positive targets. Scores include training and testing pixels, and show true positive, true negative, and entire image.



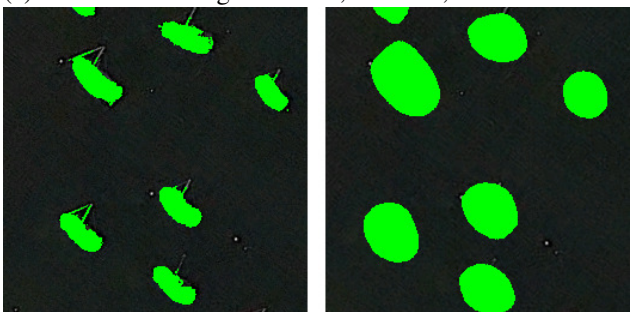
(a) Ground truth. 19901 positive pixels.



(b) No Offset: + 94.5%, - 96.5%, tot 95.5%



(c) Block Processing: + 97.4%, - 90.6%, tot 94.0%



(d) Details: (left) No Offset, (right) Block Proc.

Fig. 5. Boat output images. Scores are true positive, true negative, and total for entire image.

(a) Ground truth. 7874 positive pixels.

(b) No Offset: + 97.5%, - 96.4%, tot 97.0%

(b) Block Processing: + 98.3%, - 87.7%, tot 93.0%

(c) Details: (left) No Offset, (right) Block Processing

Fig. 6. Face output images. Scores are true positive, true negative, and total for entire image.

Together, the spatial languages (Complete, No Offset) are the overall top performers, meaning that spatial operators are useful for many images studied. Curiously, even though the Complete language is a super-set of the No Offset language, it was not as good a performer in many instances. This contradicts conventional wisdom that GP evolution will select the best language operators for a problem at hand. In our experience, the more complex language was not refined by evolution. This may be due to the larger language defining too complex a search space. On the other hand, the Raw Features and Block Processing languages do not use spatial operators. It is clear that the block processing strategy we used with the Block Processor is advantageous for that language, as it is the main technical difference between it and the Raw Features language, which is the poorest performing language of the four studied.

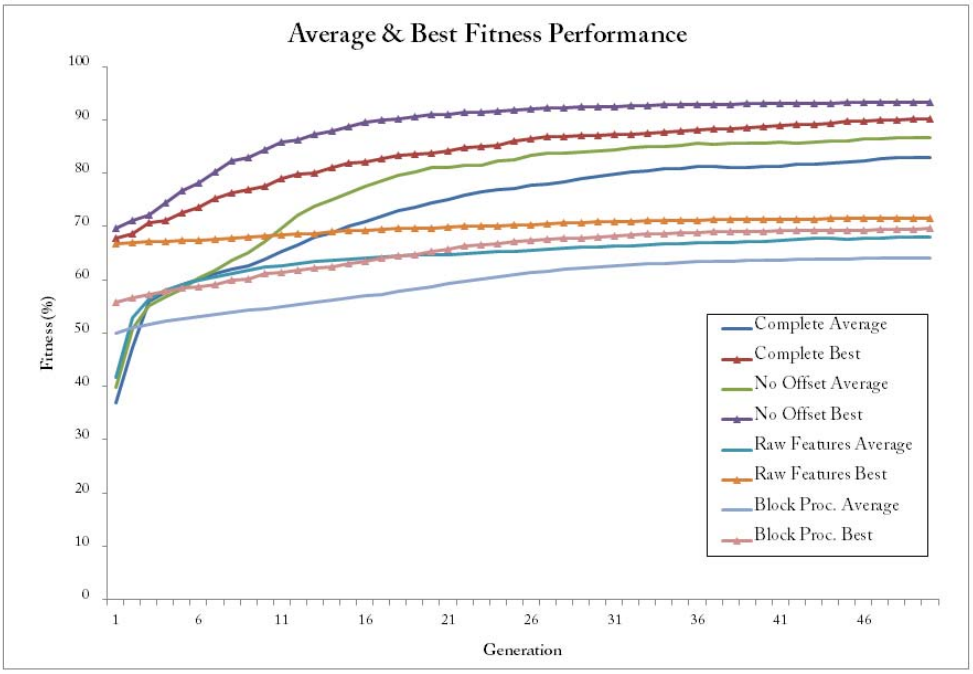


Fig. 7. Fitness performance graph for Texture 1 runs (average 20 runs). Average is measured for entire population, while best is the single best scorer in the population. Ideal performance (ground truth image) is 100%.

Figure 4 show some texture image results of the experiments. Images (a-d) are the best solution image results for Texture 1 recognition, and (e-h) for Texture 2. Green indicates positive identification. Therefore, the high density of green in the Texture 1 area (see Figure 1) that is apparent in images (a-d) corresponds to the high true positive scores. Green in other texture areas denotes false positives. Hence, one can see where the Complete, No Offset, and Block Processing languages were tricked by the texture in the bottom-left corner. Similarly, the Raw Features language was fairly liberal in (c) when identifying positive instances, resulting in a low overall score of 65.6%.

Texture 2 is one of the more difficult textures to recognize. The results in (e-h) show that it is usually recognized, but more mistakes (false positives and negatives) arise in (e,f,g). The Block Processing results in (h) have a high degree of false positive. A good result of the Block Processing language on Texture 3 is shown in (i).

Also note that inter-texture boundaries can be difficult for some languages. This is because boundaries represent complex mixes of different textures. The spatial operators are free to overlay across these boundaries, which can complicate recognition of positive regions. We consider boundary artifacts to be acceptable noise.

Figure 5 show sample results for the aerial boat image. The No Offset result in (b) is very close to the ground truth (a), and has an 88.05% image score. False positive are mostly found in the ground clutter on the right-side. For comparison, a Block Processing result is shown in (c). The details in (d) show the primary advantage of pixel-based classification over block processing — the ability to do precise classification.

Figure 6 show some results with the group photo. The faces are identified, but with quite a lot of false positive results on arms and background. The block result shown is much worse. In hindsight, this is a challenging image to analyze, as the faces are very difficult to ascertain without higher-level image processing. Although the group photo's results are worse than the textures and boats, they are useful in that they highlight some limitations of the languages when applied to difficult images.

Figure 7 shows the fitness performance for the Texture 1 runs. The top 4 curves are for the spatial languages, which show superior fitness to the non-spatial languages.

VI. Discussion

Our results show that pixel-classification languages with spatial operators are preferable to those without spatial operators, and to block-classification languages. As described in Section III-A, our block processing strategy used to analyze the results in Table VI is not the manner in which block languages were originally used. In [2][3], an overall classification score (percent) for an entire image would be determined after applying multiple block classifications to it. However, by using exhaustive block overlays and thresholding, we could more accurately compare the block language performance with pixel-based languages.

Examining research in [2][3][5][6], we note that the GP expressions used for block processing were apparently not applied in a random-sampled manner to images. Rather, fixed coordinate positions for block overlays were used on training images, which might make training less effective than if random-sampled coordinates are used. Computational performance in wall-clock speed is also advantaged by the ability to classify large regions of images at once.

Our experience is that using a block classifier with random sampling of images during training and testing is more challenging for block classification. Since the simple block language samples only a relatively sparse number of pixel points in an image region, it is difficult to learn pattern

concepts for complex images when too sparse a set of training points are used. Spatial languages overcome this due to their ability to extract pertinent features over an image region, and in our case, using average and standard deviation calculations (which are akin to blur and edge filters). Furthermore, because a pixel-based spatial language can error for one pixel rather than an entire region, errors are far less costly.

VII. Conclusion

Our results show that spatial pixel classifiers were the best performers for the problems we studied. The rich set of spatial operators used means that they are capable of sophisticated feature extraction. We found that the block classifier was competitive with the pixel classifiers in a few images tested. Our thresholding approach to block language processing showed that the block classifier language studied has very respectable capabilities, considering the simplicity and efficiency of the language used. It should be noted that later research in [5][6] applied block languages in a manner closer in spirit to our approach.

There are many directions for future investigations. The GP vision literature (see Section I) shows that much more complex GP languages for pattern classification are possible. The language definitions we used could be made considerably more advanced, by considering sophisticated image processing primitives. The degree to which complex languages are required, however, depends upon the application. Song *et al.*'s work shows that a very rudimentary language is capable of texture recognition, and there is no reason to believe that all problems require complex primitives. Future work should also consider alternate kinds of vision applications. New problems will undoubtedly require the use of appropriate pattern classification languages.

Another direction of research is the application of GP-evolved classification languages towards image segmentation. Our "window sweeping" of classifiers on the aerial boat image produced interesting results, and especially so with the block classifier language. We intend to explore this application further, and are considering implementation with NVIDIA CUDA to speed up processing [17].

Our motivation here is to study the effect of feature extraction language within the problem domain of GP vision. We did not consider other computer vision paradigms in this research. There are many other computational approaches to pattern recognition in the computer vision literature, for example, computational intelligence techniques (neural networks, clustering) and mainstream computer vision algorithms. Future research should consider a detailed comparative study of GP vision technology with other computer vision approaches in the literature.

Acknowledgment

Thanks to Cale Fairchild for his assistance with systems issues. This research is partially supported by NSERC DG 138467.

References

[1] P. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 40, no. 2, pp. 121–144, 2010.

[2] A. Song, T. Loveard, and V. Ciesielski, "Towards genetic programming for texture classification," in *AI 2001: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, M. Stumptner, D. Corbett, and M. Brooks, Eds. Springer Berlin Heidelberg, 2001, vol. 2256, pp. 461–472. [Online]. Available: http://dx.doi.org/10.1007/3-540-45656-2_40

[3] A. Song, "Texture Classification: a Genetic Programming Approach," Ph.D. dissertation, RMIT University, April 2003.

[4] R. Poli, "Genetic programming for feature detection and image segmentation," in *Evolutionary Computing*, ser. Lecture Notes in Computer Science, T. Fogarty, Ed. Springer Berlin Heidelberg, 1996, vol. 1143, pp. 110–125. [Online]. Available: http://dx.doi.org/10.1007/BFb0032777

[5] A. Song and V. Ciesielski, "Fast texture segmentation using genetic programming," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, vol. 3, 2003, pp. 2126–2133 Vol.3.

[6] ——, "Texture analysis by genetic programming," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2, 2004, pp. 2092–2099 Vol.2.

[7] D. Howard, S. C. Roberts, and R. Brankin, "Target detection in sar imagery by genetic programming," *Adv. Eng. Softw.*, vol. 30, no. 5, pp. 303–311, May 1999. [Online]. Available: http://dx.doi.org/10.1016/S0965-9978(98)00093-3

[8] N. Harvey, S. Perkins, S. Brumby, J. Theiler, R. Porter, A. Cody Young, A. Varghese, J. Szymanski, and J. Bloch, "Finding golf courses: The ultra high tech approach," in *Real-World Applications of Evolutionary Computing*, ser. Lecture Notes in Computer Science, S. Cagnoni, Ed. Springer Berlin Heidelberg, 2000, vol. 1803, pp. 54–64. [Online]. Available: http://dx.doi.org/10.1007/3-540-45561-2_6

[9] W. A. Tackett, "Genetic programming for feature discovery and image discrimination," in *Proceedings of the 5th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 303–311. [Online]. Available: http://dl.acm.org/citation.cfm?id=645513.657740

[10] J. F. Winkeler and B. Manjunath, "Genetic programming for object detection," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*. Morgan Kaufmann, 1997, pp. 330–335.

[11] B. Ross, A. Gualtieri, F. Fueten, and P. Budkewitsch, "Hyperspectral Image Analysis Using Genetic Programming," *Applied Soft Computing*, vol. 5, no. 2, pp. 147–156, 2005.

[12] R. Poli, "Genetic programming for image analysis," in *Proceedings of the First Annual Conference on Genetic Programming*. MIT Press, 1996, pp. 363–368.

[13] M. Ebner, "A real-time evolutionary object recognition system," in *Proceedings of the 12th European Conference on Genetic Programming*, ser. EuroGP '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 268–279. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01181-8_23

[14] ——, "Towards automated learning of object detectors," in *Proceedings of the 2010 international conference on Applications of Evolutionary Computation - Volume Part I*, ser. EvoApplicatons'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 231–240. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12239-2_24

[15] P. Brodatz, *Textures: a photographic album for artists and designers*. Dover New York, 1966, vol. 66.

[16] D. Montana, "Strongly Typed Genetic Programming," *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[17] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[18] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 2, 2001, pp. 1070–1077 vol. 2.

[19] S. Luke, "Ecj," last accessed June 4, 2014. [Online]. Available: http://cs.gmu.edu/~eclab/projects/ecj/