The Unconstrained Automated Generation of Cell Image Features for Medical Diagnosis

Taras Kowaliw
ISC-PIF, CNRS
57-59 rue Lhomond, Paris 75010, France
taras@kowaliw.ca

Wolfgang Banzhaf
Memorial University of Newfoundland
St. John's, NL, A1B X3X, Canada
banzhaf@mun.ca

ABSTRACT

An extension to a non-linear offline method for generating features for image recognition is introduced. It aims at generating low-level features automatically when provided with some arbitrary image database. First, a general representation of prioritized pixel- neighbourhoods is described. Next, genetic programming is used to specify functions on those representations. The result is a set of transformations on the space of grayscale images. These transforms are utilized as a step in a classification process, and evolved in an evolutionary algorithm. The technique is shown to match the efficiency of the state-of-the-art on a medical image classification task. Further, the approach is shown to self-select an appropriate solution structure and complexity. Finally, we show that competitive co-evolution is a viable means of combating over-fitting. It is concluded that the technique generally shows good promise for the creation of novel image features in situations where pixel-level features are complex or unknown, such as medical images.

Categories and Subject Descriptors

I.5.4 [**Computing Methodologies**]: Pattern Recognition—*Computer Vision*; 1.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search*

General Terms

Algorithms, Design, Experimentation

Keywords

genetic programming, muscular dystrophy, image processing, pattern recognition, co-evolution, TEF

1. INTRODUCTION

In this paper, we explore an approach to the automatic generation of features for use in image classification problems. Rather than applying only a general set of predefined features, or devoting significant human effort to the careful design of database-specific features, we instead use machine learning to extract new features from a given image database. We do so by finding transforms on the raw pixel-space of a provided image database. These transforms consist of graphs derived via genetic programming (GP) defined on local pixel neighbourhoods. In this sense, they may be viewed as a generalization of convolution masks. These transforms temporarily increase the dimensionality of the space, ideally in ways which emphasize important features of the database. Next, a simple statistical description of each transformed image is collected, and the resulting vectors are used to train a classifier.

Overall, this approach can be considered feature extraction, since the system accepts a high-dimensional data source as input (an image database) and outputs a low-dimensional description. However, unlike many feature extraction techniques, ours is highly non-linear, requiring a great deal of processing to be undertaken by the transforms, leaving little work left for the classifier.

The advantages of this approach are four-fold:

- the discovered transforms and their descriptions can be computed quickly (although they are slow to train). It takes a single pass over the image to extract the requisite feature vector;
- the transform is simply represented as a mathematical expression, one which can be easily interrogated or modified;
- we can evolve any number of distinct features in a natural way, and, as we shall demonstrate, effectively allow the system to automatically determine a good number;
- placing the burden of the classification task on this transformation stage allows for highly processed output to be viewed as images, making the evolved solutions easy to understand and present to a human analyst.

In this iteration of the work, we introduce a new and less constrained version of the system: firstly, all utilized features will be emergent from the raw pixel-space, suggesting greater applicability to poorly understood image databases; secondly, the number of features generated is under the system's control, allowed to vary via specially-designed genetic operators; thirdly, the neighbourhood cardinality is also under the system's fine-grained control, varying as a component of the representation; finally, a competitive co-

GECCO'12, July 7–11, 2012, Philadelphia, Pennsylvania, USA.

evolutionary mechanism is utilized to control the over-fitting that results with a naïve evolutionary algorithm.

We will show that our new approach, under co-evolution, is capable of improving on our previous results, and matching the current state-of-the-art. Further, it does so with far less pre-processing (*i.e.*, no pre-defined features) than in all previous work. Also, we will show that the unconstrained system makes similar or better choices than the previous structured solution, demonstrating the capacity of the system to automatically determine appropriate solution structure and complexity.

1.1 Review

Many forms of classification, especially image classification, require a low-dimensional collection of features for processing. Feature extraction refers to the extraction of a lower-dimensional collection of features from a database, ideally gaining parsimony but not sacrificing capacity for image processing. Commonly, a measure of effectiveness, such as the Fisher criterion, is used to evaluate the efficacy of a feature set, allowing for an analytic approach to the problem via, say, PCA, LDA, or variants.

Here instead we concentrate on a wrapper approach, where a collection of features is evaluated via their performance in a classifier directly. The advantages to a wrapper approach include the allowance of any sorts of features (*i.e.*, no assumptions regarding resulting data distributions), including features consisting of non-linear combinations of source data. The downside to the use of a wrapper-based approach is that: (a) runs take a great deal of time to execute, since a classifier must be trained for each evaluation; and (b) features tend to be useful for the classifier used during training, but not necessarily useful with other classifiers. This capacity for non-linear combinations creates a greater range of potential outputs, and even occasionally leads to situations where dimensionality *increase* might improve efficacy. For this reason, we prefer to refer to these techniques as *feature creation*.

Feature creation is often undertaken via evolutionary approaches, typically by taking a collection of pre-defined features and utilizing a GP process to define a new collection of re-combined features. It is believed that GP is well suited to cases where the interrelationship between variables is poorly understood, or when the size or shape of the ultimate solution is unknown [19]. A recent review lists successes in all stages of the classification process, due largely to the flexibility of the technique and the capacity for insertion of heuristics or hybrid techniques, but at the expense of computational expense for training and comprehensibility of solutions [4].

GP and similar evolutionary algorithms are often used in image processing tasks in particular, where dimensionality is especially critical, and the capacity to extract mathematical descriptions is thus desirable [5, 9, 13, 15, 23, 24, 25]. Flexible representations are highly desirable in several particular areas, especially those in which techniques inspired by human vision might be sub-optimal, for instance, in non-standard visual tasks such as satellite, multispectral, hyperspectral, or medical imagery [8, 21, 18]. We are here particularly interested in Cartesian Genetic Programming (CGP) [16], a form of GP which has shown success in naturally producing parsimonious (*i.e.,* more human-comprehensible) solutions. CGP has been applied successfully to image processing in several contexts [20, 22, 23].

Transform-based Evolvable Features (TEFs) were first introduced by Kowaliw, Banzhaf, *et al.* [12]. In that previous work, a single evolved TEF was evolved and added to a collection of pre-defined image features, applied to a medical cell task (the same task we explore again here). A later application of the system explored the creation of features for the recognition of artistic style, this time utilizing entirely evolved features [14]. A key difference between this approach and most others is that TEFs are extracted directly from the raw pixel-space of the images, rather than operating on a space of pre-extracted features, meaning that unexpected local-based patterns can be found.

An approach with similar motivations is the work by Shirakawa, Nakano *et al.* [17, 23], where two different kinds of GP networks of image operators are applied in a network to images to produce more processed image transforms. Both their work and our own works directly on the pixel space, allowing GP to find transformations useful in highlighting regions of interest. Shirakawa *et al.*, however, apply their work to image segmentation rather than classification.

2. CELLSDB

The CellsDB database was collected by the Centre hospitalier de l'Université de Montréal (CHUM), where the causes and associated symptoms for Oculopharyngeal Muscular Dystrophy (OPMD) at the genetic and cellular level have been studied extensively [1]. Intranuclear inclusions (INIs) are tubular, about 8.5 nm in external and 3 nm in inner diameters, up to 0.25 μm in length, and converged to form tangles or palisades. Detection of INIs is expected to lead to the detection of OPMD. Detecting INIs, as opposed to other intranuclear patterns, is a difficult task requiring training for human classification. CellsDB is pre-segmented so that each image contains a single cell. The images are taken at 10x, 20x, and 40x zoom, divided into two categories associated with the presence or absence of INIs: "healthy" and "sick". CellsDB has previously been used for several image processing tasks [6, 7, 10, 12]. Here we break into two sets, randomly chosen for each experiment: 186 healthy and 200 sick cell images for training, and 200 healthy and 200 sick cell images for testing.

3. FEATURE CREATION

Our model — the evolutionary feature creator (EFC) — operates on some provided database of images, divided into a finite number of classes. The ultimate output of the system will be a collection of easily computed features well-suited for classification of the database by some particular classifier. The EFC consists of two parts: an evolutionary algorithm, and a collection of *feature extracting individuals*, or simply *individuals*. The evolutionary algorithm — through which good individuals are found — is discussed in Section 4; Here we discuss the definition of individuals.

Individuals consist of a collection of transformations over an image, a collection of moments, and a classifier. When presented with an input image, an individual executes the following process:

1. It computes a collection of transformations of the image. These transformations are based on the application of a mathematical function through a sliding

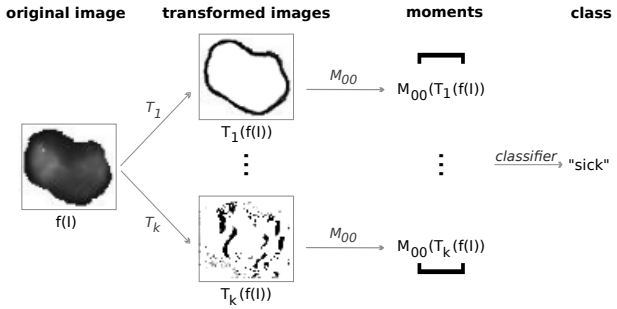


Figure 1: An overview of a feature extracting individual: First, the original image is transformed by a collection of TEFs; Next, each is converted to numerical values via the moments; Finally, that feature vector is classified.

window. The sliding window operates on a variable-cardinality neighbourhood, which we present as an improvement over the use of simple square neighbourhoods. The mathematical function is defined via a GP graph.

2. It next converts the transformed images to a feature vector via a collection of dimensionality reducing moment functions.
3. Finally, it feeds the feature vector to a classifier, which returns a class label.

This process is illustrated in Figure 1.

Our primary interest at present is in the definition and evaluation of the first item, the transformations, which we term a TEF. Our use of GP at this stage is motivated by the technique's capacity to automatically discover appropriate forms and sizes for solutions, which might allow for the accommodation of unknown image databases.

3.1 Transform-based Evolvable Features

A TEF consists of a mathematical function applied to a given image through a sliding window. Our definition deviates from previous work [12] through the usage of a new form of input neighbourhood. Otherwise, our representation of mathematical functions via CGP graphs is unchanged. First, we present our custom notion of variable-cardinality neighbourhoods, one we believe useful for many evolutionary computation representations. Next, we discuss the use of GP graphs as a means of defining general mathematical functions on those neighbourhoods.

There exist cases in evolutionary computation where two-dimensional representations are believed necessary, but it is not clear whether these cases will generalize to all two-dimensional applications. Nor is it clear that a better one-dimensional representation capable of performing the same tasks has simply not yet been found. Of course, *ceteris paribus*, one would prefer a one-dimensional representation, since: (a) the design is more amenable to existing techniques and analysis; (b) standard genetic operator design may be used; and (c) two-dimensional representations tend to grow quickly in cardinality (e.g. square mask sizes grow as n^2), or tend to have very large sets of representations of the same size. For these reasons, we believe that an intelligently chosen one-dimensional representation for neighbourhoods is a preferable option for many applications in evolutionary

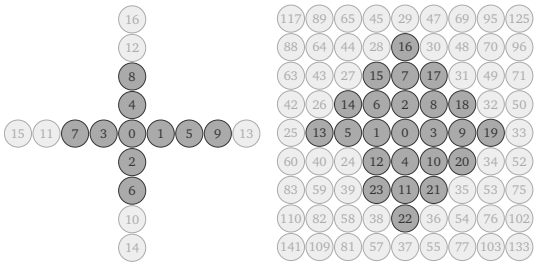


Figure 2: Neighbourhoods of type *(left)* Cross-10 and *(right)* VN-24 are highlighted.

computation. The application described in this paper — representations of pixel neighbourhoods for image processing — is one such example; a second example might be the design of a variable-length neighbourhood scheme for two-dimensional cellular automata, using neighbourhood cardinality as a measure of program complexity (as in [11]).

Here, we use variable-cardinality neighbourhoods of type *von Neumann* and *Cross*[1]. The functions are illustrated in Figure 2 and explicitly developed in Appendix A. They have been designed to give complete coverage of a two-dimensional space, or a cross-shaped subset, such that central points are given more prominence. For each neighbourhood function, given an index value n we can specify a neighbourhood of cardinality $n+1$, or select a unique point in that neighbourhood.

Once a neighbourhood type and cardinality are specified, we can present these to a CGP graph. We shall write the output of an arbitrary CGP transform T applied to a list of n values as $T(x_1, ..., x_n)$. Given some location $p \in I$, we apply T to pixel p and neighbourhood of cardinality n, denoted $\bar{p} = \{p_0, p_1, ..., p_n\}$, as follows:

$$T(f(\bar{p})) = T\left(f(p_0), f(p_1), ..., f(p_n)\right) \quad (1)$$

where $f(q) = -1$ if q is outside of the image bounds. and where if $T(f(\bar{p})) \notin [0, 1]$, we will replace it by the closest boundary value. Note that we need only query those pixels as are used by the graph, so n is a maximum neighbourhood cardinality, and, in practise, the number of inputs which need to be queried is smaller. An example of one such CGP graph can be seen in Figure 3.

Given some image $f(I)$, we can define a new image, $T(f(I))$ as follows: Let I' be an image space of the same dimensions as I. For each pixel $p' \in I'$, let $f(p') = T(f(\bar{p'}))$. Hence, every CGP graph T can be viewed as a function on the space of images. We will refer to such a transform as a TEF.

3.2 Feature Extracting Individuals

A feature extracting individual is a $(k+2)$-tuple, $I = (n, k, T_1, ..., T_k)$, where n is the maximum neighbourhood cardinality, k is the number of transforms, and T_i is a TEF. An individual is applied to an image by collecting input lists of cardinality n for each pixel in the image $f(I)$, then considering the transforms $T_1(f(I))$, ..., $T_k(f(I))$.

Once our transforms have been computed, a dimensionality-reduction step is performed. Our hope is that a good choice of moments — one well-suited to the particular database —

[1]Source code for performing these mappings is available at `http://kowaliw.ca/projects/nbhd.html`

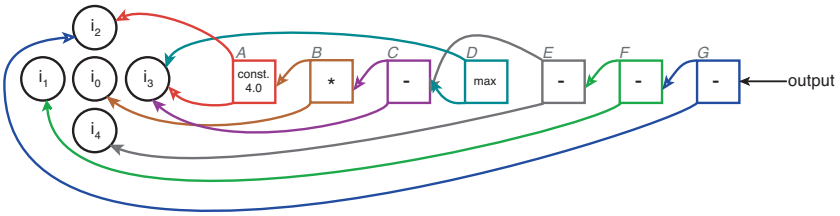


Figure 3: Example of a CGP graph with a *VN-5* neighbourhood as input (circles) and seven computation nodes (squares), *i.e.,* $n_{graph} = 7$. This graph corresponds to the Laplacian convolution mask: $output = 4 \cdot i_0 - i_1 - i_2 - i_3 - i_4$. Note the neutral code, *e.g.,* node *D* is not connected to the output.

can be used to relay meaningful information in the transformed images. Our aim is for the TEFs to find and mark useful patterns in the image source, and for the moments to return a measure of the level of incidence and location of those patterns in some particular image. In this case, we expect little value to information regarding the location of the inclusions, since it seems they appear uniformly over the cell area. Thus, we will use only the first geometric moment:

$$M_{00} = \frac{1}{|I|} \sum_{p \in I} f(p) \tag{2}$$

where $|I|$ is the size of the image, and $f(p)$ is the pixel intensity at pixel $p \in I$. Thus, following the above steps, given an image $f(I)$, an individual will return a collection of $1 \times k$ numeric values, $\{M_{00}(T_1(f(I))), ..., M_{00}(T_k(f(I)))\}$.

4. EVOLUTIONARY ALGORITHM

Our system is driven by a standard evolutionary algorithm, as described by Eiben and Smith [3]. An initial population of size $N_{initPop}$ is created. A run consists of a fixed number of generations, during which a new population of size N_{pop} is generated through:

(a) passing the best individual into the next population unchanged (single-member elitism);
(b) in some runs, selecting individuals for crossover, merger, and pruning with probability p_{cross}, p_{merge} and p_{prune} respectively;
(c) creating all other population members via mutation, using a mutation probability of p_{mut}.

Selection is accomplished through a three-member tournament. There are several additional system parameters: the graph size of the CGP transforms, n_{graph}; the maximum number of TEFs per individual, n_{TEF} (set to 50 in all cases); and the type of neighbourhood used, either *VN* or *Cross*.

4.1 Initialization and Variation

An individual can be initialized by first initializing a value for n and k randomly and uniformly, and then initializing k TEFs. As in previous work, each TEF can be represented as a list of $4 \times n_{graph}$ integers, and initialized by generating integers in the appropriate range uniformly and randomly.

These representations are varied via the familiar mutation and crossover operators. We also utilize two additional operators, merge and prune. The purpose of these operators, primarily, are to enlarge or shrink the individuals, respectively, with respect to the number of transforms k. The use of a *merge* operator is a cooperative technique — one which serves to create larger individuals out of a combination of smaller ones — previously shown to be useful in some evolutionary image classification tasks [9]. We define these genetic operators in Appendix B.

4.2 Fitness

The use of a wrapper approach to evaluation necessitates the use of a fast classifier and an off-line learning approach: in our experience, a 1-nearest neighbour (1-NN) classifier is a good choice due to its speed and efficiency. In previous work we have found that the use of this classifier forces an assumption of the use of Euclidean distance onto the resulting feature space: that is, the resulting features work best with other classifiers which use Euclidean distance as a basis, as opposed to classifiers which use other distance metrics, such as the Mahalanobis distance metric [14]. Regardless, re-evaluation through other more computationally-intensive classifiers is often successful, and sometimes provides superior performance, as is the case here with the simple logistic classifier.

Any particular individual is evaluated by converting a collection of images to feature vectors, and then evaluating a classifier's capacity to distinguish between the pattern types. The evaluation method changes between training and test runs: During training runs, the training set T is broken into two equally sized sets: one used for training proper, and the other for evaluating the trained individual. The split is randomized with each evaluation, making all evaluations stochastic. Test runs are evaluated using 10-fold cross-validation on the test set of images, V. There is no feedback from the results of the test runs to either the classifier or the evolutionary algorithm; Testing is used only to evaluate the success of the individuals on an independent set, and to track over-fitting. We evaluate our individuals using the test set every five evolutionary generations.

The classifier will return a false positive rate, FPR, for all classes $c \in C$, which we combine into the *sensitivity-specificity*:

$$SS_S = (1 - FPR(\text{“healthy”}))(1 - FPR(\text{“sick”})) \tag{3}$$

where S is the set used for evaluation, either T or V. Note that the values of SS_T and SS_V will be stochastic, due to choice of folds and randomized order of images. In our experience the variance of SS_V is quite low (< 0.01). Using SS in the definition of fitness is done since the evolutionary process is forced to find features which are useful for all classes simultaneously, which eliminates some common local minima in the search landscape.

Preceding work with CellsDB has revealed that some “sick”

examples are more easily recognized than others, and the possibility exists for a classifier to obtain a fair accuracy simply recognizing these few instances. To prevent this, we have authored a new fitness function for evaluating an individual I:

$$fit(I) = \min_{1 \leq m \leq 3} SS_{P^{(m)}}(I) \tag{4}$$

where $P^{(m)} \subset T$ is a randomized subset of the training set, with $|P^{(m)}| = 100$. This particular choice was selected from several possible forms on the basis of speed of computation.

The ultimate evaluation of an individual will involve the familiar SS_V measure, of course; To compare with SS_V for the purposes of tracking over-fitting, we also report the average training SS_T value as a more accurate measure of expected performance of the individual:

$$\langle SS_T(I) \rangle = \frac{1}{3} \sum_{1 \leq m \leq 3} SS_{P^{(m)}}(I) \tag{5}$$

We also report a more common measure of success, the *proportion* of *correct*ly classified images:

$$PC_S(I) = \frac{|\{f \in S | I \text{ correctly classifies } f\}|}{|S|} \tag{6}$$

5. EXPERIMENTS

On the basis of 200 randomly generated data points, a parameter set believed to maximize fit was selected:

$N_{initPop}$	100	N_{pop}	60
p_{cross}	0.55	p_{mut}	0.025
p_{merge}	0.035	p_{prune}	0.15
max. nbhd. card.	41	n_{graph}	200
nbhd. type	Cross		

Twenty runs using this parameter set were executed. At the end of the 100th generation, the best individual was declared the output of the system. We obtained the following results:

measure	**mean**	**s.d.**
SS_V	0.672	0.052
PC_V	0.819	0.032
n_{feat}	10.22	3.04
max. nbhd. card.	16.39	3.40

However, we found that typically the average training score, $\langle SS_T \rangle$, was approximately 0.1 larger than the test score. This large difference implies that the model is in some cases "memorizing" the training instances, or learning some other inappropriate pattern. This mimics previous work with CellsDB, where over-fitting was also an issue [12]. As such, we elected to try again using a more robust means of optimization.

5.1 Experiment with Competitive Co-Evolution

In this section, we consider the effects of using competitive co-evolution to augment the evolution of our feature-extracting individuals. Competitive co-evolution has been successfully used in evolutionary image classification in the past to eliminate over-fitting by artificially increasing the difficulty of the pattern database [13]. Here we will use co-evolution as a means of selecting samples for individual evaluation, purposefully choosing collections of images from our training set which challenge the current population. (Note that [13] went further, allowing images to "camouflage" themselves, which we will not do here for fear of interfering with presently unrecognized cell image information.)

The pattern population is a collection of lists of images, $\mathbb{P} = \{P^{(1)}, ..., P^{(N_{prey})}\}$, where N_{prey} is the size of the pattern population, and $P^{(j)} = \{i_1^{(j)}, ..., i_m^{(j)}\}$ is a list of images, $i_k^{(j)} \in T$. The idea is that each pattern individual $P^{(i)}$ is optimized to challenge the population of pattern recognizers, that is, they are evolved to select images which classifiers have difficulty recognizing. We used values of $N_{prey} = N_{pop} = 60$ and $m = 60$ for all experiments.

At the outset, each pattern individual is initialized randomly from the total set of training images: $P^{(j)} = \{urand(T), ..., urand(T)\}$. Note that it is possible for $i_k = i_l$ for some $k \neq l$. We consider this a means for evolution to specify that a particular image is important enough to gain extra weighting in evaluation. For each evaluation of a feature-extracting individual, a random pattern population member is selected and used in that individual's evaluation. Once the classifier has been computed, the fitness value is associated with the pattern individual. Let the number of times that a particular pattern individual, $P^{(j)}$, has been evaluated in a total generation be $n_E(j)$. Pattern fitness is defined as

$$fit_{patt}(P^{(j)}) = \begin{cases} 1 & ; \text{if } n_E(j) = 0 \\ \frac{1}{n_E(j)} \sum_{k=0}^{n_E(j)} fit^{(k)} & ; \text{ow.} \end{cases} \tag{7}$$

for all $fit^{(k)}$ associated with $P^{(j)}$.

At the end of every generation, we select pattern individuals for the next pattern population in a standard genetic algorithm. We use deterministic tournament selection with a tournament size of three, selecting the individual who minimizes fit_{patt}. Given two pattern individuals, $P^{(1)}$ and $P^{(2)}$, a new pattern individual is generated via *crossover*: $P^{(3)} = \{urand(i_1^{(1)}, i_1^{(2)}), ..., urand(i_m^{(1)}, i_m^{(2)})\}$. Given a single pattern individual $P^{(1)}$, a new pattern individual can be generated via *mutation*: $P^{(2)} = \{urand(i_1^{(1)}, urand(T)), ..., urand(i_m^{(1)}, urand(T))\}$. That is, mutation re-initializes each point with probability 0.5. This large value was chosen so as to prevent over-specialization by the pattern population. Otherwise, all evolutionary parameters were set to be identical to the parameters of the main evolutionary run.

5.2 Co-evolutionary Results

We ran twenty trials of the co-evolutionary model using the same parameter set as for the original experiments. We obtained the following results:

measure	**mean**	**s.d.**
SS_V	0.730	0.054
PC_V	0.854	0.031
n_{feat}	5.420	2.11
max. nbhd. card.	16.60	4.74

Welch's t-test shows that the hypothesis that the co-evolutionary model has better expected SS_V than the non-co-evolutionary model is valid ($p < 0.003$), as illustrated in Figure 4. The smaller value for n_{feat} suggests that a less complex solution has been found, implying that the co-evolution has indeed compensated for a too-complex initial model.

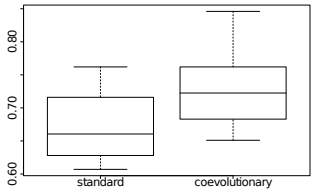


Figure 4: Comparison of SS_V values between the standard and co-evolutionary runs.

Since our features are general purpose, there is no need to use the same classifier for evaluation as was used as a wrapper during evaluation. After some experimentation, we found that superior classification results were obtained via a simple logistic classifier:

measure	mean	s.d.
SS_V (simple logistic)	**0.773**	0.057
PC_V (simple logistic)	**0.896**	0.041

5.3 Best Discovered Individual

The best individual overall was extracted from one of the co-evolutionary runs. Evaluated via a simple logistic classifier, this individual attains a $SS_V = 0.865$, encompassing true positive rates of 0.973 and 0.890, or a $PC_V = 0.922$. Table 1 illustrates this individual's transforms and their actions on some sample images. Note that the original individual had five transforms, two of which were excluded due to triviality. These features were further ordered by their rank as determined by information gain, such that T_0 is the most significant.

The T_0 feature recovers a similar strategy to the one found in previous evolutionary work with CellsDB: a selection for variation between pixels close together, and a simultaneous rejection of pixels of different intensity too far apart. This strategy detects inclusions of a range of sizes against a dark background. The similarities to the transform discovered in previous work are striking: firstly, both utilize a neighbourhood of width 6 pixels and height 5 pixels (although this is represented more parsimoniously in the new representation); secondly, both mathematical expressions are dominated by a maximum operator, contrasting the left-right and up-down directions.

The T_1 feature recovers a partial measure of the circumference of the cell, and the T_2 feature recovers a loose measure of the area of the cell. Interestingly, comparing to previous work [12], these two features correspond to the two pre-defined features included in previous evolutionary work which were, by information gain, ranked as most useful. Indeed, it appears that the unconstrained evolution of a variable number of features has re-created not only the evolved feature from previous work, but has also found improvements on the most useful two pre-defined features. Further, evolution has also recovered the appropriate neighbourhood size as was discovered in previous work (adjusting for different neighbourhood types), this time using a less constrained neighbourhood representation. Thus, we conclude that the system has successfully self-selected an appropriate solution structure and complexity for the problem at hand.

5.4 Comparison to Previous Work

In earlier work, CellsDB was evaluated using both a collection of pre-defined cell features, and those same features augmented by a single evolved transform under a collection of statistical descriptors. In that work, the predefined features alone attained an SS_V score of 0.580, and the addition of a single evolved TEF attained a mean performance of $SS_V = 0.676$ (s.d. 0.052) as evaluated by a 1-NN classifier. The maximum performance was $SS_V = 0.801$, achieved by evaluating the best features with a J48 Decision Tree[2] [12]. This new work is approximately an 8% improvement in expected performance. We should point out that this was accomplished using entirely evolved features, whereas the previous approach included several predefined cell-specific features. Moreover, the best discovered individual attained $SS_V = 0.865$, also approximately an 8% improvement in SS over previous maximal performance.

Further work has been undertaken by Guo *et al.* [6, 7], who report the proportion correct on a test set. Their technique consisted of initially pre-processing images into a histogram region of interest (HROIT), then calculating a collection of common features over the HROIT: entropy, density measures, moments, and more. Initially, the success rate is computed on the raw features unsing an SVM. Next, a hybrid genetic programming and expectation-maximization process is used to construct a single composite feature, which is similarly evaluated and found to have better efficacy. It should be noted that the test set utilized by Guo *et al.* is larger than that used here (500 versus 400 cell images, due to the choice of train-test split).

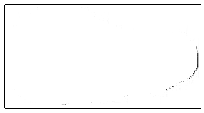
Below, we summarize the results from four approaches in terms of the PC_V measure: firstly, from the use of predefined cell based features under the best explored choice of classifier, 5-NN, extracted from [12]; secondly, from the use of an SVM to classify pre-defined features based on a HROIT, extracted from [7]; thirdly, from the use of the hybrid GP-EM algorithm to classify those same HROIT-based features, also extracted from [7]; finally, the present technique based on evolved TEF features and a simple logistic classifier:

	max	mean	s.d.	#runs
cell + 5-NN	*NA*	0.775	0.015	40
HROIT + SVM	0.810	0.786	0.041	10
HROIT + GPEM	0.920	**0.902**	0.035	10
TEF + logistic	**0.922**	0.896	0.041	20

The work by Guo *et al.* and the present approach are both clearly superior to naïve techniques on pre-defined features. The mean outcome of that approach is slightly higher than the present technique, but within error. The maximal value of our technique is highest, but very close to the value obtained by Guo *et al.*. We conclude that the present technique is of approximately equal merit as the previous state-of-the-art on CellsDB.

[2]Note that some data inconsistencies existed in the CellsDB database at the time that this work was undertaken, and that since, a few images have been re-categorized. This will not change test scores by more than 1%.

Table 1: Visualization of the best discovered individual; *(left)* the TEFs; *(right)* the action of the TEF on samples from CellsDB.

TEF	expression	healthy sample	healthy sample	sick sample	sick sample
original	Id				
T_0	$\max\left\{243.97^{(i_9-i_0)}, \max\left\{\frac{i_{15}}{i_4}, i_{10}\right\}\right\}$				
T_1	$\frac{i_3^{i_{12}}}{i_2 i_4}$				
T_2	$thresh(i_{11}, i_5^{i_3})^{\sqrt{i_{15}}}$				

6. CONCLUSIONS

We have improved over previous results on a medical cell classification task, and matched the results of the current state-of-the-art. This new work is also distinguished by being significantly less constrained than our previous work and most other existing approaches: all features emerge from the raw pixel-space, and the system self-selects an appropriate solution structure and complexity. In conjunction with evidence obtained from previous experiments with another poorly understood image database [14], we present our evolutionary feature creator as a viable method of automatically generating features from raw pixel spaces, especially in contexts in which low-level patterns are complex or poorly understood.

7. ACKNOWLEDGMENTS

Our thanks to CHUM and Mr. Tarundeep Dhot for providing the CellsDB images. Some classifiers and metrics are implemented via Weka [26]. W.B. was partially funded by an NSERC discovery grant (RGPIN 283304-07).

8. REFERENCES

[1] B. Brais, J. Bouchard, Y. Xie, D. Rochefort, N. Chretien, F. Tome, R. Lafreniere, J. Rommens, E. Uyama, and O. Nohira. Short GCG expansions in the PABP2 gene cause oculopharyngeal muscular dystrophy. *Nature Genetics*, 18:164–167, 1998.

[2] A. Browder. *Mathematical Analysis: An Introduction*. Springer-Verlag, 1996.

[3] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[4] P. Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *Trans. Sys. Man Cyber Part C*, 40(2):121–144, 2010.

[5] H. Guo, L. Jack, and A. Nandi. Feature generation using genetic programming with application to fault classification. *IEEE Trans. on Systems, Man and Cybernetics — Part B: Cybernetics*, 35(1):89–99, 2005.

[6] P.-F. Guo and P. Bhattacharya. An evolutionary approach to feature function generation in application to biomedical image patterns. In *11th conference on Genetic and evolutionary computation (GECCO)*, pages 1883–1884, 2009.

[7] P.-F. Guo, P. Bhattacharya, and N. Kharma. An efficient image pattern recognition system using an evolutionary search strategy. In *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 599–604, 2009.

[8] D. Howard, S. Roberts, and C. Ryan. Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recogn. Lett.*, 27(11):1275–1288, 2006.

[9] N. Kharma, T. Kowaliw, E. Clement, C. Jensen, A. Youssef, and J. Yao. Project CellNet: Evolving an autonomous pattern recognizer. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(6):1039–1056, 2004.

[10] N. Kharma, H. Moghnieh, J. Yao, Y. Guo, A. Abu-Baker, J. Laganiere, G. Rouleau, and M. Cheriet. Automatic segmentation of cells from microscopic imagery using ellipse detection. *IET Image Processing*, 1(1):39–47, 2007.

[11] T. Kowaliw. Measures of complexity for artificial embryogeny. In *10th annual conference on Genetic and evolutionary computation (GECCO)*, pages 843–850, 2008.

[12] T. Kowaliw, W. Banzhaf, N. Kharma, and S. Harding. Evolving novel image features using genetic programming-based image transforms. In A. Tyrrell, editor, *IEEE Congress on Evolutionary Computation (CEC)*, pages 2502–2507, 2009.

[13] T. Kowaliw, N. Kharma, C. Jensen, H. Moghnieh, and J. Yao. Using competetive co-evolution to evolve better pattern recognizers. *International Journal of Computational Intelligence and Applications*, 5-3:305–320, 2005.

[14] T. Kowaliw, J. McCormack, and A. Dorin. Evolutionary automated recognition and characterization of an individual's artistic style. In *IEEE Congress on Evolutionary Computation (CEC)*, 2010.

[15] K. Krawiec, D. Howard, and M. Zhang. Overview of object detection and image analysis by means of genetic programming techniques. *Frontiers in the Convergence of Bioscience and Information Technologies*, 0:779–784, 2007.

[16] J. Miller. Cartesian genetic programming. In J. F. Miller,

editor, *Cartesian Genetic Programming*, Natural Computing Series, chapter 2, pages 17–34. Springer, 2011.

[17] Y. Nakano, S. Shirakawa, N. Yata, and T. Nagao. Automatic construction of image transformation algorithms using feature based genetic image network. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2010.

[18] R. Poli. Genetic programming for image analysis. In *1st Conference on Genetic Programming (GECCO)*, pages 363–368, 1996.

[19] R. Poli, W. Langdon, and N. McPhee. *A Field Guide To Genetic Programming*. Lulu Enterprises, 2008.

[20] P. Rosin and J. Hervas. Image thresholding for landslide detection by genetic programming. In L. Bruzzone and P. Smits, editors, *Analysis of multi-temporal remote sensing images,*, pages 65–72. World Scientific, 2002.

[21] B. Ross, A. Gualtieri, F. Fueten, and P. Budkewitsch. Hyperspectral image analysis using genetic programming. *Applied Soft Computing*, 5(2):147–156, 2005.

[22] L. Sekanina, S. Harding, W. Banzhaf, and T. Kowaliw. Image processing and CGP. In J. Miller, editor, *Cartesian Genetic Programming*, Natural Computing Series, pages 181–215. Springer Berlin Heidelberg, 2011.

[23] S. Shirakawa, S. Nakayama, and T. Nagao. Genetic image network for image classification. In *Applications of Evolutionary Computing*, volume 5484 of *LNCS*, pages 395–404. Springer-Verlag, 2009.

[24] A. Song and V. Ciesielski. Texture segmentation by genetic programming. *Evolutionary Computation*, 16(4):461–481, 2008.

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

[26] H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufman, 2nd edition, 2005.

APPENDIX

A. NEIGHBOURHOOD FUNCTIONS

The *von Neumann* neighbourhood is meant as a generalization of the two-dimensional cellular automaton neighbourhood of the same name, designed by modifying a diagonalizing function normally used in set theory [2]. It is easier to first understand the inverse, *unVNify*. Given some point, (x, y), we can place it into a partition by (discrete) magnitude, $m = |x| + |y|$. The number of points of smaller magnitude is $S = 1 + 4\left(\frac{(m)(m+1)}{2}\right)$. Given this information, it suffices to index points by their x-value:

$$unVNify(x,y) = \begin{cases} 0 & ; \text{ if } x, y = 0 \\ x + m + S & ; \text{ if } y \geq 0 \\ 3m + x + 1 + S & ; \text{ ow} \end{cases} . \quad (8)$$

To obtain the function *VNify*, we perform the inverse: Given some index n, first we find the maximal $m \in \mathbf{N}$ such that $2m(m-1) < n$; For $n > 1$, this is equivalent to setting $m = \left\lceil \frac{-1+\sqrt{1+2n}}{2} \right\rceil$; Note that m is now the value of $|x|+|y|$ from the previous calculations. Next, let $n' = n - 2m(m-1) - 1$. Finally, we use the x-axis as an index to reference the point:

$$VNify(n) = \begin{cases} 0 & ; \text{ if } n = 0 \\ (n' - m, n') & ; \text{ if } n' \leq m \\ (n' - m, 2k - n') & ; \text{ if } n' \leq 2m \\ (3m - n', 2m - n') & ; \text{ if } n' \leq 3m \\ (3m - n', n' - 4m) & ; \text{ ow} \end{cases} . \quad (9)$$

The *Cross* neighbourhood is defined through the following function:

$$Cross(n) = \begin{cases} (0, (-1)^{\lfloor \frac{n \bmod 4}{2} \rfloor \lceil \frac{n}{4} \rceil}) & ; \text{ if } n \bmod 4 \text{ even} \\ ((-1)^{\lfloor \frac{n \bmod 4}{2} \rfloor \lceil \frac{n}{4} \rceil}, 0) & ; \text{ ow} \end{cases} \quad (10)$$

with inverse

$$Uncross(x,y) = \begin{cases} 4 \cdot |\max\{x,y\}| - 3 & ; \text{ if } x > 0 \\ 4 \cdot |\max\{x,y\}| - 2 & ; \text{ if } y < 0 \\ 4 \cdot |\max\{x,y\}| - 1 & ; \text{ if } x < 0 \\ 4 \cdot |\max\{x,y\}| & ; \text{ ow} \end{cases} \quad (11)$$

Although we do not use it in our experiments, for completeness we also introduce the *Moorify* function. It is convenient to think of the *Moorify* function as mapping the index to the perimeter of a square. Let $k \in \mathbf{N}$ be the largest k such that $(2k+1)^2 \leq n$, and let $S = (2k+1)^2$. S is the number of points contained in the square beneath the location to which we will map our index. We let n' be $n - S$, the index on the perimeter of the square to which we will map. We let the diameter of our current square perimeter be $d = 2(k+1) + 1$, and the radius $r = k + 1$. Then, we have:

$$Moorify(n) = \begin{cases} (n' - r, r) & ; \text{ if } n' \leq d \\ (r, r - (n' - d + 1)) & ; \text{ if } n' \leq 2d - 1 \\ (r - (n' - 2d + 2), -r) & ; \text{ if } n' \leq 3d - 2 \\ (-r, (n' - 3d + 3) - r) & ; \text{ ow} \end{cases} . \quad (12)$$

B. GENETIC OPERATORS

Merge: This operator combines two individuals, provided some conditions are met. For any two individuals, $I^{(1)}$ and $I^{(2)}$, a third child individual is created through the union of the parents' features,

$$I^{(3)} = \Big(\max\left\{ n^{(1)}, n^{(2)} \right\}, k^{(1)} + k^{(2)}, T_1^{(1)}, ..., T_{k^{(1)}}^{(1)}, T_1^{(2)}, ..., T_{k^{(2)}}^{(2)} \Big) \quad (13)$$

This third child is returned provided that it is not too large ($k^{(1)} + k^{(2)} < n_{TEF}$) and that there is an increase in fitness beyond what we would expect from the stochasticity of individual evaluation, *i.e.* if

$$fit\left(I^{(3)}\right) \geq \max\left\{ fit\left(I^{(1)}\right), fit\left(I^{(2)}\right) \right\} + 2\sigma \quad (14)$$

where σ is computed by comparing five calculations of fitness for the best individual from the previous population. If the conditions are not met, merge returns a copy of the first parent.

Prune: The *prune* operator serves to provide pressure on the population to decrease individual complexity, provided that the fitness of the pruned individual does not decrease. An individual selected for pruning, $I^{(1)}$, will have a child, $I^{(2)}$, created where the child has either its neighbourhood cardinality decreased by one, or a randomly selected TEF removed:

$I^{(2)} = copy\left(I^{(1)}\right)$
if $rand < \frac{1}{2}$ **then**
  $n^{(2)} = n^{(1)} - 1$
else
  $k^{(2)} = k^{(1)} - 1$
  delete $T^{(2)}_{urand(\{1,...,k^{(1)}\})}$ and re-index
end if

If $fit\left(I^{(2)}\right) \geq fit\left(I^{(1)}\right) + \sigma$, the prune operator returns the pruned child; Otherwise, prune returns a copy of the original parent.