Typed Cartesian Genetic Programming for Image Classification

Phil T. Cattani and Colin G. Johnson
Computing Laboratory
University of Kent
Canterbury

Abstract

This paper introduces an extension to Cartesian Genetic Programming (CGP), aimed at image classification problems. Individuals in the population consist of two layers of functions: image processing functions, and traditional mathematical functions. Information can be passed between these layers, and the final result can either be an image or a numerical value. This has been applied to image classification, by using CGP to evolve image processing algorithms for feature extraction. This paper presents results which show that these automatically extracted features can substantially increase classification accuracy on a medical problem concerned with the analysis of potentially cancerous cells.

1 Introduction

Classification in data mining refers to the process of building a model for the relationship between feature values and class membership, such that previously unseen records can be assigned class membership as accurately as possible. One particular form of classification, important for example in medical applications, is the classification of images.

Feature extraction from images can be a long and laborious process involving expert knowledge. It is not necessarily self-evident which features of an object help to discriminate between one class and another class, so expert knowledge is used to extract appropriate features. This process can be time consuming because each of these features must be either measured by hand, or a specific program must be written to extract each of the features individually. Extracting as many features as possible and adding them to a machine learning algorithm will not necessary improve classification accuracy rates; it is the quality, not the size, of the feature set which most strongly influences the accuracy rate of a machine learning algorithm [1]. Because of these difficulties, automating the feature extraction process is an important research problem.

Beginning with Section 2, we will first briefly discuss background research into Cartesian Genetic Programming and how it has hitherto been applied to image processing, image recognition, and feature extraction. We will then very briefly introduce our proposed novel method of extending the Cartesian Genetic Programming approach to feature extraction. In Section 3, we will describe in detail how our CGP program works and how it uses a novel method to extract 'useful' features. In Section 4, we outline the results of a classification experiment conducted with a set of images of cancerous and non-cancerous Pap Smear cells. Finally, in Section 5, we discuss our results, the limitations of the experiment and its implications.

2 Background

Our work is primarily based on a seminal paper by Miller and Thomson titled Cartesian Genetic Programming [6], written in 1998. This paper introduces the concept of Cartesian Genetic Programming, on which all other CGP programs are founded. The program outlined in the paper by Miller was used to solve symbolic regression problems and used a limited function set of four algebraic functions: +,-,*,div. Similar to other Genetic Programming (GP) methods, CGP uses a population of candidate programs and assesses their fitness by running them on a training data. CGP uses an indexed graph representation for the various functions involved in the programs, and the strings that make up the population represent the functions and connections between them.

1. Features which improve accuracy rates will heretofore be known as 'useful' features.

A number of previous papers have applied Genetic Programming methods to *feature construction*, that is, the combination of existing features in new ways that enhance the accuracy of the classifier. Examples of these include the work by Otero et al. [7], Smith and Bull [10], and Krawiec [3]. This is a powerful application of GP; however, it does not work with raw data, but with a set of features that have already been extracted by experts.

By contrast, the work described in this paper is focused on *feature extraction* for data sets consisting of image data. This means learning how to derive numerical values from the properties of the images. This means that we can provide the feature extraction algorithm with raw images, and automatically create a new feature set without needing expert knowledge.

A small number of papers have explored the application of GP to image transformation. Poli [8] uses GP to evolve programs, based on low-level processing of pixel values in the images, in order to derive image transformation algorithms for segmentation and image enhancement; similarly, Harding [2] uses related methods to generate noise-reduction filters. Colton and Torres [1] use a higher-level representation in which parts of the function set are themselves image processing primitives e.g. median, inverse and threshold. They apply these methods to evolve image filters, focused on the kinds of "artistic" filters found in programs such as Photoshop.

More directly relevant to our work are several other papers that use GP for feature extraction. Völk et al. [11] use CGP to extract features from images, however, their work uses pixel-level operations rather than higher-level image processing primitives. Krawiec and Bhanu [4] use Linear Genetic Programming to evolve programs comprised of compositions of image processing functions. In this paper we use a similar approach, but we use CGP and we include a layer a basic mathematical functions as well as image processing functions.

Finally, Shirakawa et al. [9] apply a CGP-like method to evolve acyclic networks consisting of image transformations, feature extraction processes and arithmetic operations. This is similar to the work presented in this paper. However, in our method we allow numerical values calculated during the program to be used as parameters for image transformation functions, rather than using a fixed sequence of types. This means that aspects of the images can be used during the program to influence the details of subsequent image transformations. Furthermore, our system allows numerical constants to be evolved directly: for example, a threshold value could be evolved.

3 Methods and Algorithms

3.1 Introduction to the Program

Our work is based on the paper *Cartesian Genetic Programming*, by Julian Miller and Peter Thomson [6]. In it they describe a novel method of implementing a Genetic Programming where a program is represented as an indexed graph. The encoding for Miller's program is a linear sequence of integer numbers. These numbers are grouped logically into sets of four numbers. Each set of these four numbers pertains to one node in the graph. The first three numbers represent the three input nodes, while the fourth number represent the node function. This list of node connections and node functions is the Genotype, which is then translated to an indexed graph and executed as a program.

In the paper written by Miller and Thomson, they use an indexed graph with three rows and four columns as an example. In our work, we use a graph with the number of columns set by a parameter (with a default length of 20) and two rows.

Furthermore, our indexed graph differs in that each row is *typed*: the first row of modules contains only image functions, called *Image Modules*, and the second row contains only mathematical functions, called *Math Modules*. An *Image Function* is a function which returns an Image when executed. A *Math Function* is a function which returns a number (Java Double) when executed. Either type of module can accept a combination of Image or Double as *inputs* in theory. The row which contains the Image Modules shall herein be referred to as the *Image Platform*; whereas the row which contains the Math Modules shall be known as the *Math Platform*. The two Platforms together constitute a *Station*.

An example Image Module is one which we have named *AddToImageXAmount*. This module accepts an Image as its first input, and a Java Double as its second input. It takes the value of the second input and adds this amount to every pixel in the Image passed in the first input.

An example Math Module is one which we have named *MeanPixel*. This module accepts an Image as its first input and discards its second

input. It determines the mean intensity value of all the pixels in the image and returns this as its output as a Double.

Any node in the program may accept inputs from any node(s) which are in a column position preceding the current node.

The first module on both the Image and Math Platforms are *Starter Modules*. Starter Modules simply return a constant when its function is called using the *compute()* function. In the case of the Image Platform, the Starter Module returns an image. In the case of the Math Platform, the Start Module returns a numeric constant (default value is 2). If an experimenter is performing image classification, then the program will recursively set the image constant of the Starter Module to each image in the data set. The CGP program currently accepts 8-bit grayscale images as input.

When the program is run, a method called *collectValues()* is called on each of the nodes in each platform, starting with the Starter Module (node 0), and then proceeding to the next node, and so on until the final node is reached. The *collectValues()* method contains instructions to collect the values it needs to compute the output of the primary function of that module. It collects these values by determining which nodes on the graph are input nodes for that Module. It then calls the *compute()* function on each of the modules which are located at that node. These modules then return a value (image or number) which are then used as input values for the current module. The answer to the current module is now available using the *compute()* method.

The final value of the Station is determined by calling the *compute()* method on the last node on either the Image Platform or Math Platform, depending on the task at hand. If classification is the objective, then the compute function is called on the last node of the Math Platform, where this numerical value is stored and associated with the Image from the dataset currently stored in the Image Starter Module. These values then form a feature for that set of images.

This process is summarised in figure 1.

3.2 Typed Cartestian Genetic Programming

Our Typed CGP program contains two sets of modules; one set of Image Modules and one set of Math Modules. These sets are interchangeable with other sets. The Image Modules currently use three different interfaces. These are the following:

- First input value is an Image, second input value is a Double, output value is an Image
- First input value is an Image, second input value is an Image, output value is an Image
- First input value is an Image, second input value is ignored, output values is an Image.

The Math Modules current also use three different interfaces. They are the following:

- First input value is Image, second input value is a Double, output value is a Double.
- First input value is Image, second input value is ignored, output value is a Double.
- First input value is a Double, second input value is a Double, ouput value is a Double.

3.3 The Fitness Function

The usefulness of any individual genome encoding is determined by how it scores on the fitness function being used at the time.

The default fitness function which we used for our program is called Fisher's Ratio [5]. The program was built to be able to plug and play fitness test modules. The second fitness function module which we used in our test is the entropy test (see e.g. [12]). We implemented our own version of the entropy test which we describe below.

3.3.1 The Fisher's Ratio

The Fisher's Ratio is the ratio of the square of the difference in mean values between two groups divided by the sum of the variances of two groups:

$$\text{Fisher's Ratio} = \frac{(m1-m2)^2}{v1+v2}$$

This statistically estimates the separateness of the two distributions. A low value indicates poor separateness, whilst a high value indicates that the two distributions are well separated.

3.3.2 The Entropy ratio

Entropy is a measure of the disorder or uncertainty in any system. For a system which potentially contains two classes, the entropy value is 0 when there is no uncertainty about the probability of the next value being of a certain class.

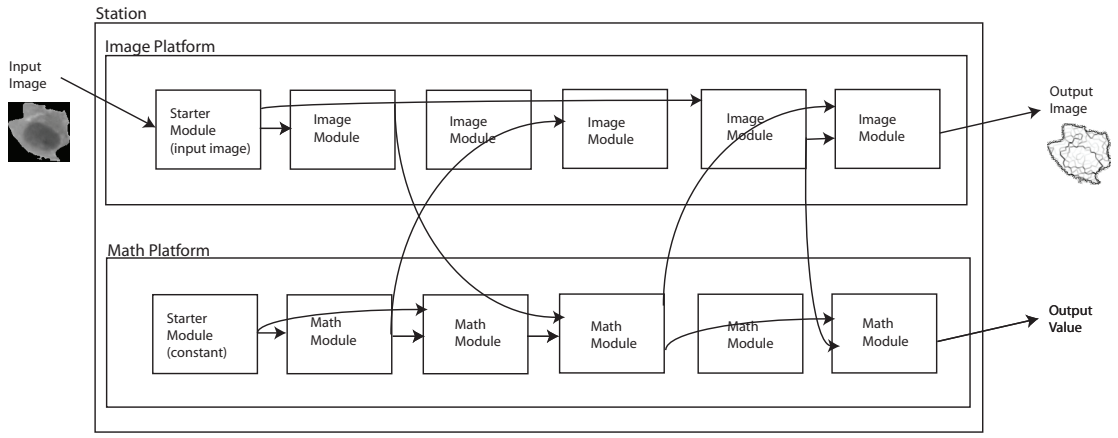

Figure 1: Example of a candidate program.

This would occur in a set of items which is comprised entirely of one class. Entropy would reach it's maximum value of 1 when there is maximum uncertainty about the probability of the next value being of a certain class. This would occur in a set of items where there is an equal distribution of items from both classes in the predicted classes.

The formula we used to calculate Entropy in our Entropy Fitness Test module was:

$$\text{Entropy} = (-(p_1)\log_2(p_1)) - ((p_2)\log_2(p_2))$$

Where the p_1 and p_2 are the probabilities of the first and second class respectively appearing in the first half of a list of sorted values for the images of both classes.

In other words, the feature values associated with each image and the known image class is stored in an array of size N. This array is then sorted according to feature value and precisely half of the array is then discarded to form a new array with size $N/2$. This new array contains the $N/2$ largest feature values.

The probability p_1 is equal to the ratio of images from one class to total number of images in the new array. While the probability p_2 is 1 - p_1. The Entropy value is then calculated using these two probabilities[2].

3.4 Overall Process

In this section we have described the various components of the Typed CGP system for image classification. We now describe how these are put together.

The initial population of size n consists of a randomly generated set of strings of digits, each of which defines a Station. This population is then evolved by a standard evolutionary algorithm process: i.e. we calculate the fitness of each member of the population, select the best b members of the population, and form a new population by first making a direct copy of these b members and then filling the remainder of the population with mutants of these members. This process is then iterated many times, until either x generations have passed or a fitness threshold has been exceeded. Features are stored whenever the fitness both exceeds a minimum threshold and exceeds the current best fitness value.

In the experiments below, $n = 20$, $b = 3$, $x = 100000$. This is typical for CGP, which uses small population sizes, high selection pressure and a large number of generations.

4 Results

In our first test, we chose two groups of different type of cells from a database of images of Pap Smear cells. The Pap-Smear database used for this test is a set of 917 images which were scanned at The Department of Pathology at the Herlev University Hospital (Patalogisk Anatomisk Institut). Twenty features were originally manually extracted and the cells classified by two cyto-clinicians. It has been made available on the web for public use for research

2. This implementation of Entropy will only work in the special case where the size of the number of images from each class is equal.

Class	Type	Class Name	Number of Images	Subtotal
Group 1	Normal	Normal Superficial	74	
Group 2	Normal	Normal Intermediate	70	
Group 3	Normal	Normal Columnar	98	242
Group 4	Abnormal	Light Dysplastic	182	
Group 5	Abnormal	Moderate Dysplastic	146	
Group 6	Abnormal	Severe Dysplastic	197	
Group 7	Abnormal	Carcinoma in Situ	150	675

Table 1: The Pap Smear Data Set

	Original 20 features		Orig. + 3 Fisher		Orig. + 1 Entropy		Orig. + All 4	
Classifier	No f.s.	F.s.	No f.s.	F.s.	No f.s.	F.s.	No f.s.	F.s.
NaiveBayes	68.33%	73.33%	73.33%	78.88%	70.00%	75.00%	75.55%	82.22%
MLP	82.77%	73.33%	84.44%	76.66%	82.22%	80.00%	90.00%	86.11%
SMO	81.11%	76.66%	81.11%	82.22%	82.22%	78.88%	80.55%	80.00%
IBK1	71.11%	71.66%	78.33%	83.33%	71.66%	74.44%	78.33%	82.77%
IBK2	72.77%	71.11%	77.22%	80.00%	73.33%	74.44%	77.22%	78.33%
IBK3	74.44%	76.66%	82.77%	83.88%	73.33%	78.33%	81.66%	83.88%
IBK4	74.44%	78.33%	80.55%	79.44%	73.33%	80.55%	80.55%	80.55%
J48	72.22%	75.55%	80.55%	78.88%	77.22%	78.88%	81.66%	79.44%
JRIP	69.44%	68.88%	74.44%	76.11%	78.33%	73.88%	78.88%	76.66%

Table 2: Results for the four experiments. Column 1 describes the classifier used. The remaining columns consist of two columns per experiment, the first without feature selection (No f.s), the second with (F.s.). The four experiments are: the original 20 features alone (columns 2 and 3); the original 20 features plus 3 features generated using the Fisher test (columns 4 and 5); the original 20 features plus 1 feature generated by the Entropy test (columns 6 and 7); and, finally, the original 20 features plus all 4 of the generated features (columns 8 and 9). Highest accuracy for each experiment is highlighted in grey.

into automated classification systems.[3] The database divides the cell images into seven different classes; three classes for *normal* cells and four classes in total for three levels of dysplasia and carcinoma in situ. We chose 180 images (the first 90 images from each class) in total from groups 3 and 6 from the database. Group 3 is the set of 'Normal Columnar' cells which are classified as *Normal*. Group 6 is the set of 'Severe Dysplastic' cells which are classified as *Abnormal* (see table 1).

The reason for this choice is that we wanted to see how well the system could differentiate between normal and abnormal cells. Specifically, groups 3 and 6 were chosen because group 3 has a similar cellular morphology to group 6, which means the groups are difficult to distinguish. A non-expert human would find it difficult or impossible to differentiate between the two groups of cells.

Our setup consists of the following: The program we use for classification is the open source data-mining software Weka (http://www.cs.waikato.ac.nz/ml/weka/) We use a varied set of classification algorithms included in the Weka program. These are: MLP, SMO, IBK (using with nearest neighbour setting set of [1..4]), J48 and JRIP. For an explanation of each of these algorithms, see the Weka documentation in the program.

Each test is run with a different set of features. The baseline test uses only the original 20 features measured by human experts. Each subsequent test uses a combination of the original features and newly extracted features. The newly extracted features were selected by the CGP program as being the most fit according to either the Fisher's test or an entropy test. Which one is used for that particular test is indicated in the title.

For each test run, we run two trials: one without using any feature selection algorithm, another with the CSFSubsetEval feature selection algorithm included in Weka using the default parameters. Our data set is the collection

3. http://fuzzy.iau.dtu.dk/smear/download.html

of 180 cell images as described above.

The results (classification accuracies) are given in Table 2.

5 Discussion

The results using the image dataset of Pap Smear cells are encouraging. The program was able to generate new features using the Fisher's test fitness test which immediately improved the maximum accuracy rate across our set of classifiers. Interestingly, the Entropy test did not yield any features which improved maximum accuracy when combined with the original set of 20 features. However, when the two sets of newly generated features were combined with the original 20, accuracy rates improved dramatically across all of the classifiers. The maximum accuracy rate went from 82.77% with the original 20 features to 90% with the combined feature set, thus demonstrating the quality of the generated attributes.

More classification experiments need to be done with large image sets. This could include data sets that already have hand-extracted features, and data sets that do not.

The Typed CGP program allows an experimenter to plug and play different fitness test modules. Because the nature of the fitness test modules is that they only test for the 'usefulness' of a feature in isolation, using this modular capability to generate features using different fitness test modules is more likely to improve accuracy rates than using only one fitness test module. Using several fitness test modules is likely to produce a complimentary features set.

In the future, we hope to alter our program to maintain a database of extracted features so that more sophisticated fitness test algorithms can be incorporated into fitness modules which take into account the potentially complimentary aspects of newly extracted features.

References

[1] Simon Colton and Pedro Torres. Evolving approximate image filters. In *Applications of Evolutionary Computing, EvoWorkshops 2009*, volume LNCS 5484, pages 467–477, 2009.

[2] Simon Harding. Evolution of image filters on graphics processor units using cartesian genetic programming. In *2008 IEEE Congress on Evolutionary Computation*, 2008.

[3] Krzysztof Krawiec. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines*, 3(4):329–344, 2002.

[4] Krzysztof Krawiec. Visual learning by evolutionary feature synthesis. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 376–383, 2003.

[5] H. Lohninger. *Teach Me Data Analysis*. Springer, 1999.

[6] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In *Genetic Programming: Proceedings of the 2000 European Conference on Genetic Programming*, pages 121–131. Springer, 2000. LNCS, Vol. 1802.

[7] Fernando E.B. Otero, Monique M.S. Silva, Alex A. Freitas, and Julio C. Nievola. Genetic programming for attribute construction in data mining. In *Genetic Programming: Proceedings of the 2003 European Conference on Genetic Programming*, pages 384–393. Springer, 2003. LNCS 2610.

[8] Ricardo Poli. Genetic programming for feature detection and image segmentation. In *Selected Papers from AISB Workshop on Evolutionary Computing. Lecture Notes in Computer Science*, volume 1143, pages 110–125. Springer, 1996.

[9] Shinichi Shirakawa, Shiro Nakayama, and Tomoharu Nagao. Genetic image network for image classification. In *Applications of Evolutionary Computing: EvoWorkshops 2009*, volume LNCS 5484, pages 395–404, 2009.

[10] Matthew G. Smith and Larry Bull. Using genetic progrmming for feature creation with a genetic algorithm feature selector. In *Proc. Parallel Problem Solving From Nature (PPSN-2004)*, volume LNCS 3242, pages 1163–1171, 2004.

[11] Katharina Volk, Julian F. Miller, and Stephen L. Smith. Multiple network cgp for the classification of mammograms. In *Applications of Evolutionary Computing, EvoWorkshops 2009*, pages 405–413, 2009. LNCS 5484.

[12] Sholom M. Weiss and Nitin Indurkhya. *Predictive data mining: a practical guide*. Morgan Kaufmann, 1998.