Fast Image Classification by Evolving Partial Solutions

Olly Oechsle and Adrian F. Clark

VASE Lab, Computing and Electronic Systems
University of Essex, Colchester, CO4 3SQ, UK
{ooechs,alien}@essex.ac.uk

Abstract This paper presents a scheme for improving the speed of evolving classifiers by genetic programming. It works by evolving partial solutions, each capable of distinguishing a single class. The technique is demonstrated on several problems in image classification and its speed of convergence is compared with conventional genetic programming.

1 Introduction

Research has been conducted into computer vision for roughly 50 years, yet the 'holy grail' of a generic artificial vision system still eludes us — and is likely to do so for some years to come. Nevertheless, there have been some impressive feats in sub-disciplines such as inspection, surveillance and 3D reconstruction. The authors' research is motivated by the wish to develop generic capabilities that can be tailored to specific vision tasks purely through training. In this context, Genetic Programming (GP) is attractive because it is the only machine learning technique currently available that is able to accept as input pixels straight from a camera and generate as output features, labels, *etc.*

Developing image processing operators and systems by GP is slow for several reasons. Firstly, the vast variability of natural images means that training involves a significant number of images if robustness is to be obtained, and the fitness function must evaluate each individual in each generation against the entire training set. Secondly, image processing operators typically involve processing regions rather than single pixels, and this is slow if the operators are themselves obtained by GP. And finally, despite its generality and flexibility, GP converges to a solution comparatively slowly. The major contribution of this paper is a scheme that makes possible more rapid evolution of solutions where more than one 'answer' is possible.

The context for the research reported herein is image classification, wherein regions or features in images are assigned to one of several categories. The particular aim was to generate vision systems that are able to cope with a larger number of classes than previous GP-based classification work; a brief summary of such work is presented in Section 2.1. Its development was motivated by AdaBoost [**?**], which improves the performance of a set of weak classifiers discovered by a separate learning algorithm. That technique has been used to great effect in, for example, face recognition using only the sums and differences of the pixels of rectangular image regions [**?**]. Although AdaBoost could be applied to GP, the fact that it involves multiple training runs would make the overall process very slow.

The 'partial solution' technique presented essentially works by increasing the rate of convergence. Existing mechanisms for achieving this are reviewed in Section 2.2, while the technique itself is described in Section 3. Two disparate image classification tasks — simple shape recognition and automatic number plate recognition (ANPR) — are described in Section 4 and the performance of the 'partial solution' approach is compared with other GP tools. Finally, Section 5 presents conclusions and outlines further work.

2 Previous Work

2.1 Classification by Genetic Programming

The first application of GP in the computer vision domain appears to be the work by Tackett, who used GP to evolve an automatic recognition system to recognise targets in military environments [1]. He introduced what has become a common theme in the use GP for computer vision, namely the evolution of feature-specific detectors from a set of quantities calculated from the region around a pixel; Tackett provided spatial features consisting of the means and standard deviations of regions of different sizes. The results obtained from the evolved detector were compared with those from an multi-layer perceptron neural network; it was found that GP delivered the more accurate solution.

In developing the GP-based solution to this problem, Tackett knew that a detection rate of only 96% was required, and he was able to terminate the GP when this rate was achieved. This luxury was not afforded to the neural network however, and in attempting to achieve a 100% success rate it is likely that it actually over-fitted (*i.e.,* fitted features in the noise) and this contributed to its ultimately poorer performance.

Johnson, Maes and Darrell [2] used GP to locate the positions of right and left hands on a silhouette image of a person. Instead of using pixel statistics, they used a basic 'point' data structure storing (x, y) coordinates. Important points around the silhouette, such as the corners of the bounding box, the centroid and the right-most point, were made available as terminals; and other functions were provided that could perform operations on two such points, such as adding them together or finding the mid??-point between them. Their results were quite successful — their best program could find 93% of left hands, although the programs evolved performed less well on validation data.

Winkeler and Manjunath [3] demonstrated the use of GP for face detection using a multi-scale windowed approach. Within each window, some fifty-two pixel-based features were made available to the GP system, as well as other image processing features such as Gabor features. The window was then passed over the image, at each point making a decision whether it contained a face or not. By repeating the process with different scales of window, the system could identify faces of different sizes. A good individual solution cited by the authors had a tree size of over 3,000 nodes which introduced a severe performance bottleneck. In a second experiment, a segmentation program was developed to distinguish roughly the areas that contained faces from areas that did not. When combined into a multi-stage approach, the processing cost was reduced by 75% and the false negative rate was also cut significantly (although the effect on the detection rate was not reported).

Roberts and Howard [4] used GP to recognise vehicles in aerial images. Single-pixel-thickness circular statistics, invariant against rotation and noise, were used as features. Four different diameters were used, with the average intensity, standard deviation, edge count and an edge distribution measure calculated for each. The authors used a bias ratio in their fitness function to determine the sensitivity/specificity ratio subjectively. The evolved detector was able to detect 89% of the vehicles with a 14% false alarm rate; however, there was no testing on unseen training data, and only relatively few images used for training so the generality of their solution could not be established.

Zhang and Ciesielski [6] were among the first to investigate the detection and classification of multiple kinds of object. They conducted several experiments with different image types: square/circle classification, coin classification, and hæmorrhage/aneurism detection in retinal images. Again implemented using a windowing approach, they used various (computationally intensive) circular and rectangular statistics. Despite a generous selection of features, a restrictive function set consisting only of arithmetic operators was used. Post-processing was used to ensure that only the centre of each object was detected and that there multiple detections of the same object did not arise.

Song and Ciesielski [5] applied GP to texture classification and segmentation, a task which often concerns vision researchers but is not often investigated using GP. They showed that GP was able to outperform the C4.5 Decision Tree algorithm [**?**] marginally when differentiating each texture from 47 others. However, the simplicity of the patterns involved meant that perfect scores could be attained by both methods, largely invalidating any conclusions drawn.

2.2 Improving the Efficiency of Genetic Programming

Arguably the simplest way to refine the performance of GP is to 'tune' the values of the numeric values that appear in individual programs in each generation [**?**]. This can be done randomly, using a genetic algorithm [**?**], or using a numeric scheme such as conjugate gradients. While effective for optimizing the values of these constants, it has little effect on the overall direction of evolution.

Montana [**?**] introduced the concept of strongly-typed genetic programming, a technique that imposes extra constraints on the tree-building and learning algorithms to improve the viability of programs by ensuring that functions only operate on appropriate data types. Montana demonstrated that the inclusion of strong typing increases the likelihood of generating legal programs. Solutions to a variety of problems involving vector and matrix manipulation were presented, and that in each case the constraints imposed by strong typing decreased the search time or improved the generalization performance of evolved solutions.

Roberts [**?**] demonstrated theat caching the results of certain trees, so that they would not require recalculation, increased the efficiency of GP. He observed that, due to the nature of crossover, some sub-trees found in generation N will be found again in generation $N + 1$ and will require recalculation. If the outputs for certain common subtrees are calculated, then the GP process would be able to proceed more quickly. His experiment involved the segmentation of multiple images, so the amount of memory required to cache the result of each subtree was substantial. Roberts proposed caching the results of evaluations to disk, provided the process of re-evaluating the result would

take longer than the time to retrieve the cache from disk. Using his method, he was able to reduce the amount of subtree evaluations by up to 66%, which in that instance translated into a time saving of 52%.

Roberts and Claridge [**?**] suggested that "choosing pixels with uniform probabilities will under-represent important parts of the image," based on the observation that certain classes will sometimes dominate in a particular image. They suggested using a reduced feature set and k-means clustering to divide the pixel training set into a number of classes, then to randomly sample an equal number of points for each class to form the final sample. This, they suggested, both ensures that each class is represented to approximately the same degree and permits a solution to be evolved using fewer points in total. However, no results were presented which quantify the benefit of this strategy and our own experiments appear to reveal no significant difference between removing points using the above method and removing them at random.

Monsieurs and Flerackers [**?**] compared several different bloat control methods on a relatively simple regression problem. They examined various strategies which can be used to reduce the size of program trees, such as code editing, where analytical processes are used to identify introns (superfluous parts of the tree) and remove them. A form of hill-climbing may also be used after crossover, which only adds new individuals to the next generation if they are either better or smaller than their parents. A linear metric or parsimony pressure can be added to the fitness function of individuals to impose an evolutionary desire to keep individuals small, or a more brutal technique can simply remove all individuals above a certain size. Monsieurs and Flerackers also introduced a new technique similar to the latter but where the cutoff size was altered dynamically to be a certain percentage higher than the tree size of the best individual. This keeps a constraint on the size of the individuals but is not arbitrary, allowing programs to grow to the natural size at which good solutions are most plentiful. They conducted a number of comparative experiments and found that code editing actually has little effect on the size of trees. Hill-climbing does significantly reduce bloat, though at a slight performance cost. Dynamic size limiting was shown both to keep bloat low and increase performance of the individuals.

Several of the above [**?**,**?**,**?**] all reported the benefit of splitting the detector into two or more stages, with the results from a "rough" but fast detector being fed to slower but more discriminating detectors. The "rough" detector is a solution trained to attain high levels of sensitivity, at the expense of lower specificity — *i.e.,* it will generate many false positives. This makes the problem easier and encourages a simpler, smaller program to be evolved. The finer detectors are expected to have both high sensitivity and specificity. As the rough detector can remove large parts of the image, there is less potential for the fine detector to make mistakes, so the accuracy of the combined detector should also improve.

3 Genetic Programming Using Partial Solutions

Standard GP aims to evolve a single program to solve a particular problem defined by a set of training data. The aim is for the program to be capable of solving all the training data, which often demands a large tree structure; however, as discussed above, GP

becomes less efficient as program size increases. Large programs often include a significant proportion of redundant code, the replacement of which through crossover and mutation wastes processor time. Perhaps more importantly, the probability that adding new features to large programs without causing some damage to the existing solution decreases as the program size increases, a feature of the so-called stability-plasticity dilemma. These two realities cause the task of the GP system to become more difficult as evolution progresses.

In this work, the authors aimed to develop a system of Genetic Programming that was less affected by each of these factors. Just as with the weak classifiers of [**?**], partial solutions are evolved that are able to solve only part of the overall problem. Here, a partial solution is defined as a program which can make correct classifications on part of the training set (see Section **??** for how useful partial solutions are identified). A number of partial solutions are combined together to form an overall solution. The scheme continues to evolve partial solutions until the overall solution is capable of a certain level of accuracy. As each partial solution is not expected to solve the whole problem, their programs tend to be small and are evolved more quickly.

Crucially, once a partial solution is identified as being useful — that is, it can solve part of the problem that has not already been solved — it is saved and *the problem is redefined to include only the remaining training data*. This removes the need for already solved data to be continuously re-evaluated, as is the case with standard GP, and also ensures that the knowledge of the partial solution is protected.

In contrast again with regular GP, partial solution(s) may be identified at any stage during the evolution of a complete solution, so an entire generation does not have to be executed before a decision is taken. This is because the criteria needed to identify a useful partial solution are not dependent on waiting until the end of the generation. Indeed, we have found that several different partial solutions may be discovered during the course of a single generation.

Each partial solution is treated as a binary classifier with respect to a particular class c. The best fitness with regard to a particular class is chosen as the partial solution's overall fitness, calculated by dividing the total number of correct classifications TP_c by the total number of training samples N added to the number of missed classifications FP_c:

$$\text{fitness} = \arg\min_c \left(\frac{\alpha \mathrm{TP}_c}{N + \beta \mathrm{FP}_c} \right) \tag{1}$$

The inclusion of the factors α and β allows the fitness measure to be adjusted to affect the individuals' sensitivity or specificity.

In order to decide whether a partial solution is useful — that is to say that it is able to solve part of the overall solution — criteria were devised, which may be summarised as follows: "Does the individual solve part of the problem that has not been solved already without making significant mistakes?" If the answer is yes, then the individual is added to the strong classifier. Specifically, four criteria must be met:

1. *Does the partial solution discriminate?* If the solution returns only one class label it is not capable of classifying. In our experience, GP often evolves this kind of lazy solution in response to training data that are weighted in favour of one particular class.

2. *Is the partial solution unique?* If the classifier returns the same results for every instance as another partial solution that has already been chosen, then this classifier is discarded as it does not add any further expertise to the solution.
3. *Does the partial solution return 'true' for data that have not yet been solved?* Each item of training data has a field indicating whether it has already been solved. Each candidate partial solution has to identify at least one instance that has not yet been solved in order to be used.
4. *Does the partial solution return 'false' for data in other classes?* If a classifier returns true for instances of one class, it should return false for all instances of other classes. However, it may mistakenly return true for other classes provided that they have already been completely solved by other weak classifiers.

The final criterion above permits a partial solution to make mistakes, provided that a previous partial solution has already solved the item of training data correctly. Hence, for the overall, combined solution, the partial solutions must be executed in the order in which they were evolved. Each partial solution is in effect a binary classifier, identifying either a particular class or returning false.

4 Experimental work

To demonstrate the effectiveness of the technique, it has been applied to a number of problems concerned with image classification. Two such problems of increasingly difficulty are presented here: shape classification (Section 4.1) and automatic number plate recognition (Section 4.2). The important point here is not that the same classification system can be trained on the problems but rather that the sub-solution approach yields either a better solution than alternatives or that it evolves it more quickly. The vision system employed in these experiments comprises three stages, illustrated in Figure **??**: the first segments features from background on the basis of colour and texture cues; the second calculates interesting features that describe 2D shape; and the final stage performs classification based on the features.

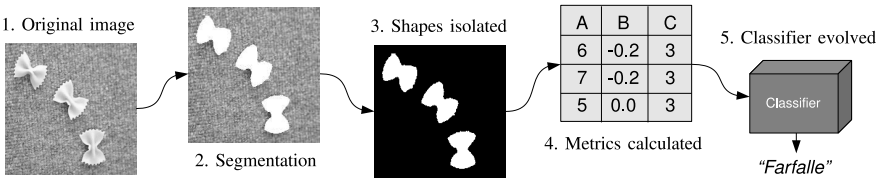

Figure 1. The vision system used for the experimental work

In the segmentation stage, a program is evolved using GP to classify each pixel in an image as being of a particular class. This is done using regular strongly-typed GP, the function set consisting of mathematical functions and logical operators (**and**, **or**, **not**) and comparison operators (**more**, **less equal**, **between**) and ephemeral random constants.

Contiguous regions of the same class are then identified, with regions associated with the 'background' class (identified as part of the training set) being discarded. In the second stage, the remaining regions are processed further to calculate some 17 metrics that describe shape, the most important of which are:

countCorners identifies the number of corners by analysing the differential radius about the shape's centre of mass

countHollows identifies the number of holes in a shape

balanceX the distance between the centre of the shape's bounding rectangle and its centre of mass in the x direction

balanceY the distance between the centre of the shape's bounding rectangle and its centre of mass in the y direction

density the proportion of the pixels in the shape that are 'solid' (*i.e.,* not part of a hole)

joints how many bifurcations the shape has, identified after skeletonization

ends the number of end-points in the shape, also found after skeletonization

The particular features calculated are, as far as possible, independent of scale and orientation, and form the input to the final, classification stage. It is here that the problem is solved using the partial solutions approach described above. The major parameters controlling evolution in the two stages are given in Table 1.

	segmentation (regular GP)	classification (partial solutions)
population	300	7500
generations	??	50
$P(\text{crossover})$	0.5	0.1
$P(\text{mutation})$	0.1	0.1
tournament size	25%	50%
maximum nodes in tree	150	20

Table 1. Parameters controlling evolution in the two stages

In the applications described below, a set of training data was created using a graphical tool that allowed a person to mark pixels as being representative of a particular class. The same training data were used to train all the GP engines. The graphical tool was also used to capture a disjoint set of test data, used to evaluate the effectiveness of the evolved classifiers.

In each case, we have evolved solutions using the authors' own GP engine with and without partial-solutions. Results from ECJ [**?**] are also given so that the reader can make comparison with a well-established GP engine. In each experiment, evolution

was repeated five times and the results are performance of the best individual across the runs are presented here. All experiments were conducted on a 2.5 GHz PC with 1 Gbyte memory, and all the software is written in Java.

4.1 Shape Recognition

This experiment concerned to recognition of five different shapes of pasta (conchigle, cavatappi, farfalle, fusilli, rigatoni). Two photographs of each type of pasta were taken at two different distances (to yield differently-sized shapes) against a relatively plain background. The training set comprised 135 different labelled examples, while the test set comprised some 150 shapes, all captured with a Logitech Quickcam Pro 9000 webcam. An example images labelled by the vision system is shown in Figure 2, while the performance of the authors' GP engine with and without the partial solutions approach is shown in Table 2 along with the figures from ECJ for comparison. **Need some comments about tree size, I think.**

Although the best solution from the authors' engine out-performs the best one from ECJ, the differences in performance are not great. However, the time taken to arrive at the solution is reduced by a factor of about 24, which definitely is significant. Hence, we are able to conclude that the partial solutions approach reduces the complexity of the classification problem so that GP is able to evolve a complete solution more rapidly.

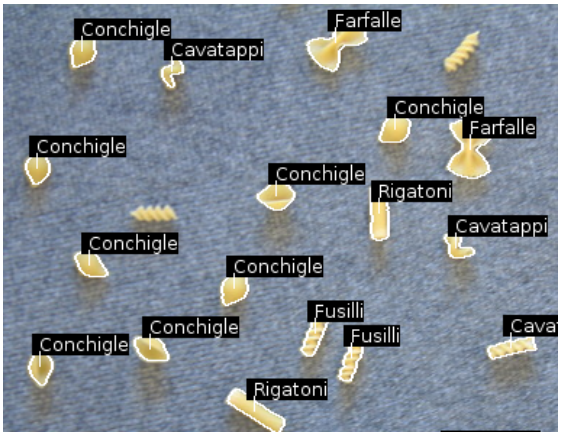

Figure 2. Image of pasta shapes labelled by the evolved vision system

4.2 Automatic Number Plate Recognition

The second problem is the reading of the characters from photographs of vehicle license plates. This is significantly more difficult because of the number of classes to be distinguished in the solution: some thirty different class labels are required. All the test

Technique	Training (%)	Test (%)	Evolution time (ms)
GP with partial solutions	100.0	90.7	4,745
GP without partial solutions	96.3	86.7	114,052
ECJ	91.1	82.7	109,000

Table 2. Performances on the pasta recognition task

Technique	Training (%)	Test (%)	Evolution time (s)
GP with partial solutions	92.0	79.5	101
GP without partial solutions	49.1	35.3	233
ECJ	??	??	??

Table 3. Performances on the number plate recognition task

images are of dark letters against a white background. Some 224 letters and numbers from 35 plates were used for training and 233 unseen characters formed the test set.

(a) Number plate image (b) Letters segmented from background (c) Individual letters identified

Figure 3. Automatic number plate recognition

Evolved solutions to this overall task typically consisted of about 110 partial solutions, evolved in about 100 seconds (Figure 3. **Do we have a labelled number plate image to include in Figure 3?** A conventional GP solution here takes only twice as long to evolve... but achieves less than half the number of correct classifications. We conclude that, for larger classification problems such as this, the 'divide and conquer' approach adopted in this work reduces the complexity of the overall problem into a series of smaller ones that are simpler to solve by GP.

5 Conclusions and Further Work

Motivated by a need to make GP reach a solution as rapidly as possible in classification problems, this paper presents a technique generates partial solution that together contribute to the overall solution of a problem. It has been shown that the technique improves convergence on image classification problems, including one that is large. Although the authors have not applied the technique to problems outside the imaging domain, they see no reason why it could not also be adapted to other problems in which there are many possible outcomes.

The GP framework used in these experiments, including the 'partial solutions' implementation and the tool used for data set creation, are available from the authors' website, which will be included in the paper if it is fortunate enough to be accepted.

References

1. Walter A. Tackett. Genetic programming for feature discovery and image discrimination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 303–309, 1993.
2. Michael P. Johnson, Pattie Maes, and Trevor Darrell. Evolving visual routines. In Rodney A. Brooks and Pattie Maes, editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 198–209. MIT Press, 1994.
3. Jay F. Winkeler and B. S. Manjunath. Genetic programming for object detection. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Proceedings of the Second Annual Conference on Genetic Programming*. Morgan-Kaufmann, 1997.
4. Simon C. Roberts and Daniel Howard. *Evolution of Vehicle Detectors for Infra-red Linescan Imagery*, volume 1596 of *Lecture notes in computer science*, pages 110–125. Springer-Verlag, May 1999.
5. Andy Song and Victor Ciesielski. Texture analysis by genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2092–2099, June 2004.
6. Mengjie Zhang and Victor Ciesielski. Genetic programming for multiple class object detection. In Norman Foo, editor, *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, pages 180–192. Springer-Verlag, December 1999.