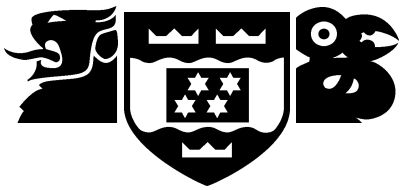
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@mcs.vuw.ac.nz

Genetic Programming for Multi-Class Object Detection

Mark Pritchard
300040217

Supervisor: Mengjie Zhang and Peter Andreae

October 19, 2002

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

Computer vision techniques are being used more frequently than every before. One of the most useful and most challenging tasks in computer vision is object detection. Object detection is the process of locating small objects within large images. This project examines the use of Genetic Programming, a learning paradigm, to solve the multi-class object detection problem. Object detection becomes multi-class object detection when there are a number of different categories of object in the image. This report looks at two sets of images one set is easy computer generated image of squares and circles, and the other is five and ten cent coins. This report presents techniques that can successfully be applied to those images to solve the multi-class object detection problem.

Contents

1 Introduction **1**
1.1 Goals 1

2 Computer Vision **3**
2.1 Feature Extraction 3
2.2 Localisation 3
2.3 Classification 4
2.4 Detection 4

3 Genetic Programming **6**
3.1 Terminals and Functions 6
3.2 Program Generation 6
3.3 Crossover 8
3.4 Mutation 9
3.5 Reproduction 9
3.6 Fitness 10
3.7 Problems (areas of concern with GP) 10

4 Tasks **13**
4.1 Easy Task 13
4.2 Coins Task 13

5 Methodology **16**
5.1 Classification Strategy 16
5.2 Terminals 16
5.2.1 Feature Extraction 17
5.2.2 Easy Images 17
5.2.3 Coins 5-10 18
5.2.4 Evaluating the Usefulness of Terminals 18
5.3 Functions 19
5.4 Clustering 20
5.5 Fitness 20

6 Results **22**
6.1 Easy 22
6.2 Coins 5 - 10 23

7 Future Work **28**
7.1 Improving Crossover and Mutation 28
7.2 Heating of Constants 28

7.3 Dynamically allocating T values 29
7.4 Sampling 29
7.5 Images 30
7.6 Terminals 30

8 Conclusion **31**

A Object Oriented Genetic Programming Package **33**

Bibliography **35**

Chapter 1

Introduction

This project examines the problem of multi-class object detection. Given an image containing many objects of different categories, the multi-class object detection problem is to identify the location and category of every object in the image. The project explores the use of Genetic Programming (GP) techniques to solve the multi-class object detection problem.

GP is an evolutionary computing process based on Darwin's theory of natural selection. The GP process is a searching/learning process that searches for a program that will solve a given problem. The programs that it searches are programs that the process itself constructs. GP constructs these programs out of a set of primitives that are specified by the user.

The GP process initially, randomly constructs a population of programs and assigns a fitness to them via a fitness function. This fitness is a measure of the effectiveness of the program at solving the problem.

The multi-class object detection problem is difficult because it involves two subproblems. The first subproblem is localisation — the process of identifying the centers of objects of interest. The second subproblem is classification — the process of classifying what class (group) an object belongs to. The detection problem is made more difficult because there are multiple classes of objects that are being classified and localised at the same time. The more different classes there are, the more difficult the classification process becomes because there are more possible outputs that the classifier can have.

The approach used in this project is to map pixel statistics from the problem domain to the inputs of the GP process. This means that the only inputs/terminals that the system uses are based on pixel statistics from the images being processed. Pixel statistics are statistics like mean or standard deviation that are applied to the raw pixels of small regions in the image and then used as inputs. The pixel statistic produced relates to the center of the region, from which it came. The benefit of using this approach is that the pixel statistics are domain independent. Having domain independent pixels statistics allows the GP process to be applied to many different object detection problems without changing the inputs that the process uses.

1.1 Goals

The key goal of this project was to show that Genetic Programming can be used to solve multi-class object detection problems. This goal can be broken into 3 sub-goals:

1. Find an appropriate function set for GP object detection.

2. Find an appropriate set of domain independent pixel statistics for object detection.

3. Find an appropriate fitness function for evaluation of a program's effectiveness at solving a detection problem where clustering is performed.

This project begins by looking at simple computer generated images that have uniform background and two object classes of interest. Then it examines the effectiveness of GP for object detection on coins. These coins were New Zealand currency 5 and 10 cent pieces.

Chapter 2

Computer Vision

Computer vision is any process that uses images as inputs, where those inputs are then processed and the results are then acted on by the computer. For example, a finger print scanner takes an image of your finger, processes it and if there is a match, it then retrieves the relevant information on that person. In this project, the computer vision techniques that are used include feature extraction, localisation, classification, and detection. Each of these techniques is described in the following sections.

2.1 Feature Extraction

Feature extraction is the process where images are converted into feature vectors. These vectors represent each pixel in the image. Often these features can involve neighbouring pixels or regions as well as the central pixel, this sort of feature is called a pixel statistic. Pixel statistics are statistics based on pixel intensities, that are centered at the center of the region. The region or input window is a fixed size and is moved one pixel at a time. Each time the input window moves a new pixel statistic is calculated, the result relates to the central pixel of the input window. An example of a pixel statistic is the mean. The mean statistic is calculated by summing the intensities of the pixels that are within the window, the sum is then divided by the number of pixels within the window and the result is one entry in the feature vector for the pixel that is at the center of the input window. This window can move across one pixel at a time and pixel statistics can be calculated for each pixel. Then the window can return to the start of the row and move down one pixel. This method is called the sweeping window method.

2.2 Localisation

Localisation is the process of finding the locations of, the centers of objects in a large image. The process of localisation does not attempt to identify what sort of object is at a specific point, rather it just identifies the position of "some" object. Localisation can be done in three different ways: before classification (identifying what the object is), after classification, or during classification. Localisation before classification would need to use some form of edge detection, then use those edges to identify some object centers. If an object was not found by localisation then the classification method would never classify the missed object. Localisation after classification would also be difficult because all possible objects have been classified which then means that the localisation process has to cluster the classifications together. This means that the following problem can occur. Suppose our classifier classifies each region center as classes 1, 2, or 3 as shown in Table 2.1.

3	2	3	2	3
2	1	2	1	2
3	2	3	2	3
2	1	2	1	2
3	2	3	2	3

Table 2.1: Classifier output

It would be very hard to tell what is a center and what not. There could be four small objects each of class 1 or just one big object of class 3.

The other concern when doing this process is that the classification process must be applied to all the possible object positions rather than just those that are actual objects. The problem is that training time can get quite significant if there are a lot of possible objects or a lot of processing is required to classify each possible object. This is a significant concern, and is explained further in Chapter 3.

2.3 Classification

Classification is the process of identifying what type of object the input is. Classification is simply a process that groups inputs into classes. It does not do any localisation, it just classifies the input. This means that it is only applied to a single object. i.e. the classifier is given a region as input and the classifier returns the class which that region belongs to.

There are a number of classification methods that can be used to classify objects in images. Some of these are:

1. Nearest neighbour: Where all training cases are stored, then the test data is compared to the training cases and classified as the same class of object as the most similar training case.

2. Nearest Centroid: Where the average value from each class's training examples are stored, then the test cases are compared to those average values for each class and classified as the class for which it is the most similar to.

3. Template matching: Where a template for each class is created, that template is then compared to each object and if the object is within some threshold it is classified as that class.

4. GP Classifier: Where the training case is used to construct a program that classifies the objects in some way. Each test case is used as input into the program that the GP system has constructed and the program outputs the class that the input belongs to.

2.4 Detection

Object detection is the process in which objects are localised and classified. There are two sorts of detection: 1) two stage detection, 2) one stage detection. Two stage detection uses the first stage to localise the objects; these localised objects are then used as inputs to the classifier. A problem occurs where the localisation process fails to locate an object— the classifier will not get a chance to attempt to classify that object. One stage detection is where both detection and localisation are applied in a single operation. This is usually done

using a sweeping window approach. The benefit of using this method is that the process gets applied to every possible object.

Chapter 3

Genetic Programming

Genetic Programming (GP) is an evolutionary computing process that was introduced by John Koza [4, 5]. It is based on Darwin's theory of natural selection that the strongest will survive. GP is an extension of Genetic Algorithms. Instead of using bit strings like Genetic Algorithms, GP uses tree structures consisting of primitives. Primitives are functions and terminals that can be used to construct tree structured programs. The essential idea of GP is that a population of programs are randomly constructed at the beginning of the process. After the initial population is constructed, each program is evaluated on a training set and given a fitness. This fitness is a measure of how well the program perform on the training set. Genetic operators are applied to a number of the programs to produce a new generation of the population that can then be evaluated. The genetic operators are crossover, mutation and reproduction. These genetic operators are discussed in detail in the following sections. The process of performing genetic operators and evaluating programs continues for a certain number of generations or until a user defined fitness level has been achieved.

3.1 Terminals and Functions

GP constructs tree structured programs that consist of terminals and functions. Terminals are inputs from the problem and functions are operations that combine terminals. For example one possible function set might be $+, -, *, /$. A terminal set might be `F1,F2,F3,F4,F5`, where `F1` is feature 1, `F2` is feature 2 and so on. One possible program that could be constructed is `(* (- F1 (+ F4 F5)) F1)`. GP must be told the number of arguments a function needs so that it can create a tree structure that has enough terminals in the right place to be a correct program. For example, if the function set included `Abs` (the absolute value) only takes one argument, the GP process should stop construction of anything like `Abs(F1 F2)`. However `Abs((+ F1 F2))` should be allowed to occur.

3.2 Program Generation

The initial generation is randomly constructed, based on population size, maximum program size and program creation method. There are two possible creation methods, *grow* and *full*[4, 5]. The full creation method requires that all programs (trees) that are generated are fully balanced. The grow method however produces unbalanced programs (trees). Another standard restriction on the initial generation is that there should be no duplication of programs. The requirement of no duplicates is not always maintained after the first generation. This means that other generations can have duplicates and there can of course be duplicates between generations.

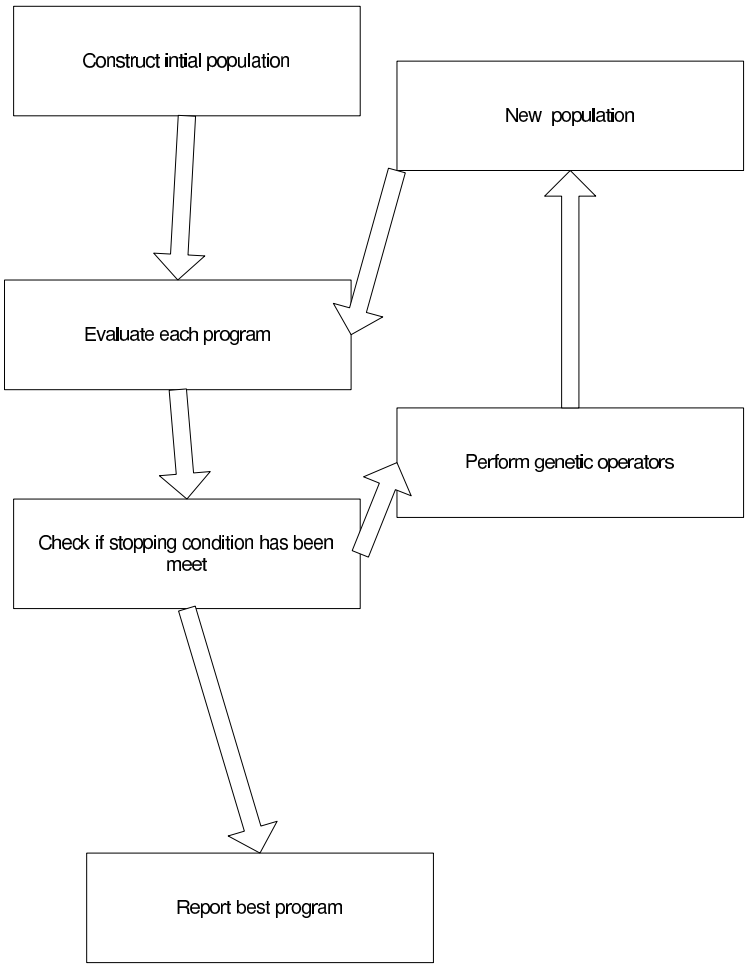


Figure 3.1: Flow chart of genetic programming process

After the initial population is produced, a fitness function is applied to each program. After the fitness has been evaluated for each program in the current generation, the three genetic operators crossover, mutation, and reproduction are applied at user defined rates. This is usually done with the user specifying the percentage of the next generation that will be generated by each of the 3 operators. Mutation and reproduction are usually applied to the best `m` and `r` programs, where `m` and `r` is the number of programs that are to be produced for the next generation by mutation and reproduction respectively. This means that the best `r` programs will be reproduced in the next generation. Reproduction is important, otherwise we can get a new generation whose best fitness is worse than the best fitness of the previous generation. Also the top `m` programs are mutated. This means that programs that are close to the best programs we have seen so far are searched. Selecting the programs to apply the crossover operator to can be done in three ways. Firstly, it can be done completely randomly. Secondly, it can be done by choosing the best `c` programs, where `c` is the number of programs to be produced for the next generation by crossover. Once the top `c` programs are selected, random selection of crossover pairs, with or without replacement can occur. Thirdly, the top $\frac{c}{2}$ programs are selected and a random partner can be selected for the program from the total population.

Once the genetic operators have been applied, their outputs are combined to make the next generation. This generation can then be evaluated, fitness value can be assigned to

each program, and the whole process can begin again, unless a stopping condition has been reached. There are generally two stopping conditions; the first is that a program has a fitness that is below the user defined fitness to stop training at; the second is that a user defined number of generations have been completed. This stopping condition is generally there just to stop training if the fitness is taking too long or is not going to get to the stopping fitness.

3.3 Crossover

Crossover is one of the operations that can be applied between generations. Crossover works by selecting two programs from the current population, and swapping a randomly selected sub-tree from each program with the other program's sub-tree (see Figure 3.2 on page 8). In the example of crossover in Figure 3.2, the two programs that the crossover operator is being applied to are P1 and P2. The two programs are selected from the current generation. Then sub-trees are chosen at random at A and B. These two sub-trees are then swapped over. The resulting programs are then placed into the new generation. In general, the programs that have better fitness are crossed over more than those with worse fitness. This means that the next generation is influenced more by the "stronger" members of the current generation.

The ideal case for crossover is when there are two programs P1 and P2, where P1 and P2 have partially correct trees. However each program has one sub-tree that is not correct for the tree but is correct for the alternative program. If these two sub-trees are swapped, then both resulting programs are correct. This is very unlikely to happen. However, what is more likely to happen is for P1 to be almost correct except for a sub-tree in P1. P2 has a sub-tree that would make P1 correct. When crossover is applied on these two programs with the correct sub-trees, P1 becomes correct but P2 could be better or worse. The worst case of crossover is when P1 and P2 are almost correct but crossover is applied on a large sub-tree of the the correct section of the programs. In this case, it is quite possible for both of the new programs to be worse than the previous programs (see Chapter 7 for more on this issue).

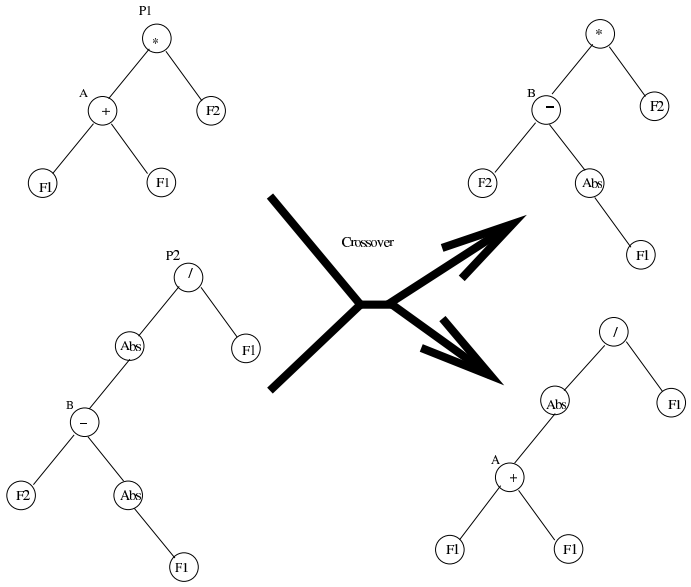


Figure 3.2: An example of crossover

3.4 Mutation

Mutation is another key operator that generates new programs. The mutation operator takes one program as input and produces one program as output. The mutation operator first selects a program from the population. Then the operator chooses a random sub-tree. This random sub-tree is then removed and a new subtree created using the same methods as program creation in the initial population. This sub-tree is then attached where the old sub-tree was deleted from. This is shown in Figure 3.3. As the figure shows, the subtree that is selected is A. A is then deleted and a new tree B is created and attached where A was. A and B can be a completely different depth and shape.

The best case for mutation is where a program P1 is almost correct except for sub-tree A. If B is the sub-tree that should be where A is, then the new program will be correct when mutation is applied. The worst case however is when A is a correct part of the program and B is not correct. In this case the new program will be worse than the old program.

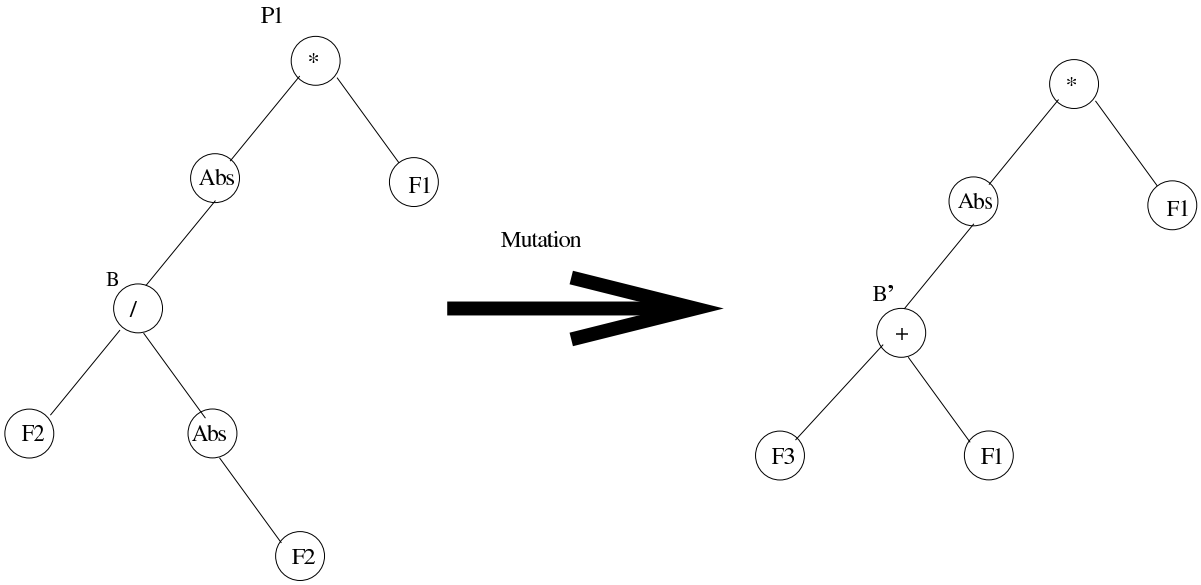


Figure 3.3: An example of mutation

3.5 Reproduction

Reproduction is the third operator applied to programs from one generation to the next. Reproduction copies a program from the current generation into the new generation. This operation does not change the program at all — it merely copies it from one generation to the next. This is applied to the top `n` percent of the population. The goal of this operator is to make sure that the best program in the new generation is at least as good as the best program in the old generation. As mentioned in the previous two sections, crossover and mutation could produce programs that are intact worse than the current generation. Therefore it is important to keep the best program(s) from the current generation in the new generation. If this did not occur, then the the GP process could produce a good program and then in the next generation make a whole population of programs that are worse than the previous best. The use of reproduction means that the best program from the newest generation is the best program that has been seen so far.

The use of reproduction however does reduce the amount of crossover and mutation that can occur in each generation as the population size must remain the same from one generation to the next. This means that if we have a population of 100 programs and we reproduce 10% of the programs, then only 90% of the programs in the new generation can be new. It is for this reason that the proportion of reproduction needs to be carefully considered.

3.6 Fitness

The fitness of a program is the measure of the effectiveness of a program at solving the problem. Fitness is often calculated as an error rate, thus the smaller the fitness, the better. As the fitness of a program affects the chances that an operator like crossover, mutation and reproduction is applied to it, the fitness is very important. What is even more important is that the fitness really does reflect the accuracy of the program at solving the problem. If a program solves the problem but the fitness reported is not zero, then the GP process will continue. It is therefore crucial that the fitness function is correct. It needs to be expressive enough to show minor improvements in the program as minor improvements in fitness and major improvements in programs as major improvements in fitness.

For object detection one possible fitness function is

$$fitness = (1 - \text{Detection Rate}) + \text{False Alarm Rate}$$

This is a basic fitness function where detection rate (DR) is the number of objects correctly detected divided by the number of desired objects. The DR can range between 0 and 1. The False Alarm Rate (FAR) is the number of objects that are incorrectly classified divided by the number of desired objects. This can range from zero to all the possible centers of windows divided by the number of objects desired. As a result of the varying ranges of the DR and FAR, constant factors are added to weight DT and FAR differently

$$fitness = a * (1 - DR) + c * FAR$$

these two constant factors are also useful for domain specific problems where DR or FAR is more important than the other. If, for example, in some problem it is very important that the DR is perfect, then the value of `a` can be increased thus putting more importance on DR. If the reverse is true and FAR is more important than DR then `c` can be increased. This fitness function is sufficient for detection problems however as shown later in this paper the function is insufficient to show minor improvements in program performance when the outputs are clustered.

3.7 Problems (areas of concern with GP)

There are a number of issues that need to be considered when using GP. They are:

1. Overfitting of training data.
2. Training time.
3. Crossover and mutation subtree selection.
4. Irrelevant program subtrees.

As with any learning system, there is a danger of overfitting the data. Overfitting the data is where a system learns a model that works very well on the training data but does not

capture the underlying model that the data comes from, and therefore performs poorly on test data. An example of overfitting can be seen in Figure 3.4 on page 12. Notice that in the figure the learnt model matches all the training data, however the function does not model the test data, this is because overfitting has occurred. Overfitting can be cased by a number things, two common causes are training for too long and not enough having enough training cases.

One of the biggest problems with the GP process is that as the number of feature vectors (or fitness cases) increases so does the training time. This is a concern because every program in every generation must be evaluated against all of the fitness cases. This means that if the population size is 500 and training runs for 200 generations, for each fitness case added there is an extra 100,000 fitness cases evaluated. Evaluation of each fitness case consists of x number of calls to the evaluate method (where x is the number of primitives in the program). This means that for each fitness case, there are $100{,}000*x$ method calls. The value of x is likely to be significant in most cases as programs are often produced with a depth 10 which could contain hundreds of primitives. This would mean that to evaluate a single fitness case for each program for each generation will take approximately 1,000,000 method calls. When there are 80,0000 fitness cases, the number of method calls rises to 80,000,000,000. This means that training can take days. There are a number of possible solutions to this including caching evaluation method results or not recalculating programs that are in the new generation if they were reproduced from the previous generation. Reproduced programs should already have a fitness associated with them so not reevaluating them is acceptable.

Mutation and crossover both select subtrees to change randomly. This is not likely to be very effective as randomly selecting subtrees is likely to affect the current program negatively. A better way of selecting subtrees would be to find a subtree or subtrees that are "bad". In this case bad means, that given the rest of the program, the subtree produces results that are not as "positive" as the rest of the subtrees, for some sense of "positive". The idea then is that subtrees that are not helping or are not good for a program are changed either by mutation or crossover. For more information on this see 7 on page 28.

The final concern with GP is that the programs that it constructs often contain redundant subtrees. For example, the program `(+ (- x x) (+ x y))` is simply `(+ x y)`. However, the GP often produces such programs. One approach is to eliminate programs with redundant subtrees between generations. However, this could have the disadvantage of causing be set of programs to converge to an overly small set of small programs. On the other hand, leaving the redundance in the programs until the final generation means that crossover is more likely to be crossing over programs that are not helpful.

These problems have been considered during this project and effort has gone into minimizing the effects of these problems.

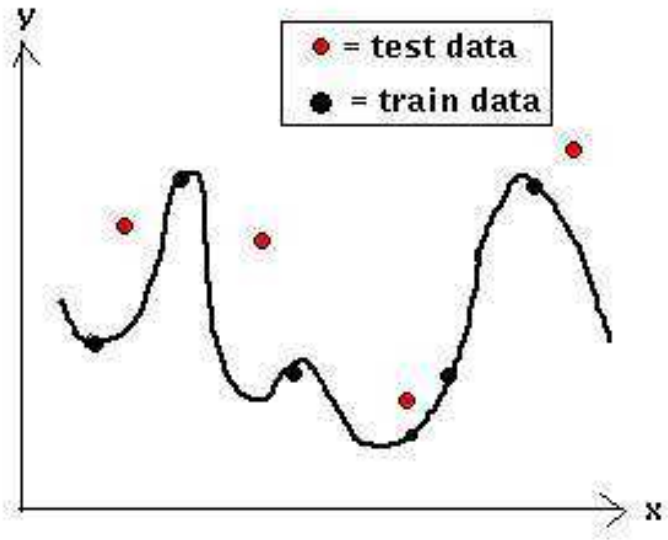


Figure 3.4: Graph of the training data and a model that is learnt from the data

Chapter 4

Tasks

Two different tasks were examined in this research. Each of the tasks was designed to test the whether the goals mentioned in the introduction were achieved. These were each of different degrees of difficulty. The first task was detecting squares and circles on a uniform background. The second task was detecting 5 and 10 cent coins on a uniform background.

When creating any of the images, two restrictions were applied. Firstly no object was allowed to overlap another object or the edge of the image. This restriction is in place because finding objects that are partially obscured is a much harder problem. Secondly, all the images used were grayscale images.

4.1 Easy Task

The first task was detecting squares and circles on a uniform background. The squares and circles were the same size. The input window was also the same size as the squares and circles. An example of the images created can be seen in Figure 4.1. The image is 400 x 400 pixels in size. The squares are 11 x 11 and the circles have a diameter of 11. The input window size is also 11. This means that there are $(400 - 10)^2 = 152,100$ centers that need to be classified correctly to solve the problem. There were 9 circles and 11 squares in each image. Thus 20 of the 152,100 possible centers should be classified as objects. The squares had an average pixel intensity of 200 and the circles an average of 150. The objects themselves were not entirely uniform with the squares having a standard deviation of 10 and the circles a standard deviation of 5. The background was completely uniform with an intensity of 100. This task was set merely to show that the GP system was working correctly.

4.2 Coins Task

The second task, which was considered more difficult, was to detect 5 and 10 cent pieces on a relatively uniform background. Initially this set of images were created using a digital camera. This caused problems due to the reflective nature of coins. The images produced using the digital camera were very sub-standard. For this reason the images were produced using a scanner. These images were scanned at high resolution and then reduced to low resolution grayscale images. The reduction in resolution reduced a New Zealand 5 cent piece to a diameter of 16 pixels. The reason for reducing the quality was that the more pixels in the image the more centers that need to be considered for each program at each generation. This is an important consideration when training can consist of a population of 500 programs and can continue for 200 generations. Consider an image that is 200 x 200 and is reduced to 100 x 100. If we train with a population of 500 for 200 generations and an image size of 200

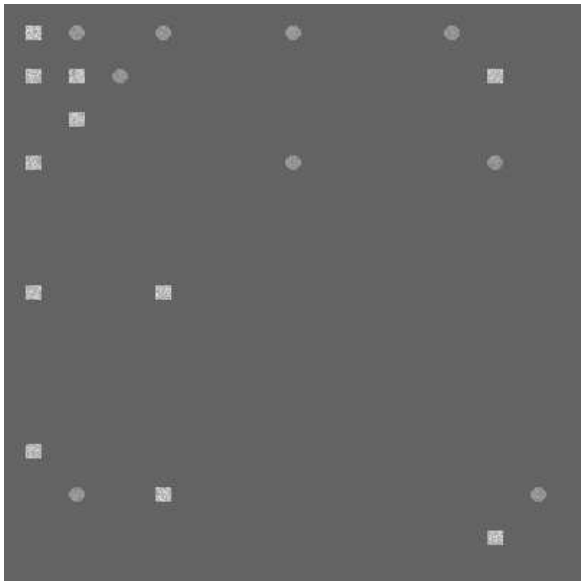


Figure 4.1: An example of the easy images

x 200. Then the calculation to work out which class a pixel is will be run approximately 500 $*$ 200 $*$ 200 $*$ 200 = 4,000,000,000 times. However if we train with an image size of 100 x 100. The calculation is run only approximately 500 $*$ 200 $*$ 100 $*$ 100 = 1,000,000,000 times. This means that training will be approximately four times faster for the smaller image. This however did make the problem harder as after making the reduction the coins were so unclear that it was almost impossible to determine whether the coin was head or tail side up.

Each image had four 5 cent pieces and four 10 cent pieces. The background was not entirely uniform as can be seen from Figure 4.2. As can be seen from the example in Figure 4.2, each of the coins looked slightly different with some pixels in the coins that have high intensity and some with low intensity. This task was set to show that the system is domain independent, that is to say that the system can go from easy images where it was detecting squares and circles and then it can be used on coins and the system will still work.

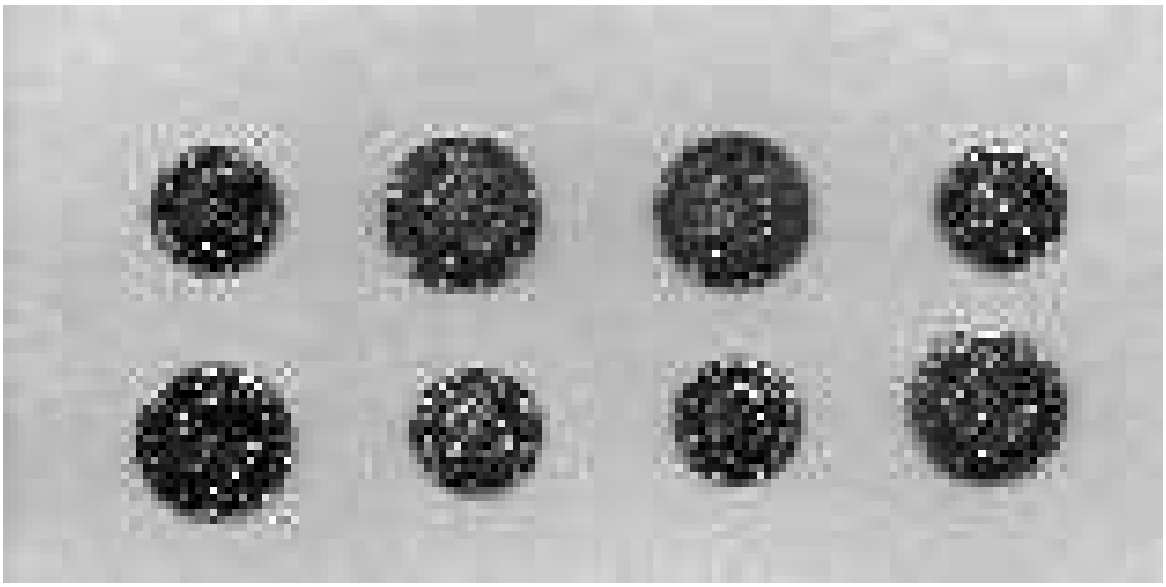


Figure 4.2: An example of the coin images

Chapter 5

Methodology

This project uses a GP system [1]. The inputs to the system are created so that the system uses a sweeping window approach to detection as described in section 2.4 on page 4. The majority of this project is based around four aspects. The four aspects are terminals, functions, fitness function and clustering. Terminals are the features that are extracted from the input images and are used as variables in the evolved programs. Functions are the operators that are used in the evolved programs. The fitness function measures the fitness of each program and drives the evolutionary process. Clustering is the process of grouping classification outputs together. These four aspects are examined in Sections 5.2 to 5.4. Section 5.1 presents the classification strategy that was used in this project.

5.1 Classification Strategy

Outputs from the GP system are floating point numbers. These floating point numbers must be mapped to classes, this mapping is called a classification strategy. The classification strategy used for this project is based on the strategy used in [9]. Where objects are classified into ranges of outputs as shown in Figure 5.1. The strategy uses a user defined constant T that is the range outputs can have for each class of object. This means that if an output value is $\frac{1}{2}T$ for example, that means that the object is class 1. If the output was $T+1$ then it would be class 2. Two important things to note about this approach are:

- The range for background object is from 0 to $-\infty$ this means that it has a bigger range than the other classes of objects. The same applies for class n where the range is from $(n-1)$ to ∞.
- There is no difference between an object being classified as $T+1$ and an object being classified as $T+\frac{1}{2}T$. This means that the classification of an object is either correct or wrong.

5.2 Terminals

Terminals are crucial in the GP process as they are the only input from the image. As a result, they need to be selected carefully, such that they are expressive enough to be able to distinguish between classes of objects. However the set of terminals needs to be kept as small as possible otherwise the search space becomes gigantic. The key then is to select the

[1] The Genetic Programming package that was used for this project was the Object Oriented Genetic Programming package. For information about the package, see appendix A on page 33.

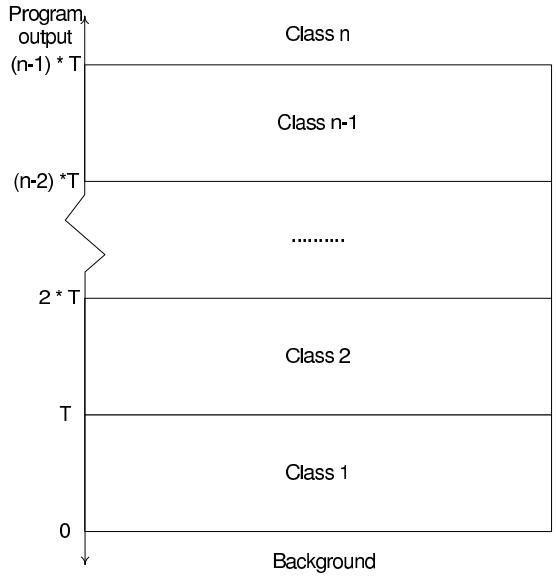


Figure 5.1: Classification Strategy

terminals that are very expressive and do not overlap over each other's expressiveness. For example if some terminal is very good at expressing the size of the object, then it would be much better to have another terminal that is expressive in terms of shape instead of size. This however is significantly harder than first thought. This is because the role of a terminal is very difficult to interpret.

5.2.1 Feature Extraction

Features were extracted using the sliding window (or sliding input window) approach. One of the problems with the sliding window approach is that features cannot be extracted from the input image for the pixel center (0,0). This is because for the window to be centered at (0,0) the window would need to start at $(-\frac{1}{2} * windowsize, \frac{-1}{2} * windowsize)$. This is clearly impossible so the features are only extracted from the centers that are at least $\frac{1}{2} * windowsize$ from the edge of the image, as shown in Figure 5.2. This restriction means that this process is not able to detect objects that are centered outside of that border.

The sliding window approach is used to extract pixel statistics at each input window position. These statistics were based on the pixels within the input window. An example of a pixel statistic is, the mean pixel intensity of the input window. This approach can improve training by reducing the search space. Instead of having 20 * 20 terminals (one for each pixel), there are only 2-10 terminals (statistics on the input window). At the same time rotational invariance can be achieved by using some statistics like mean. These statistics must be chosen carefully.

5.2.2 Easy Images

To begin with for the easy images the only statistics used were the *mean, standard deviation, first moment, second moment* and T. These were calculated on the whole input window. The first moment was calculated based on the Euclidean distance from the center of the input window. $f(x_i, y_i)$ is the pixel intensity at pixel (x_i, y_i). $d(x_i, y_i)$ is the Euclidean distance that (x_i, y_i) is from the center of the region.

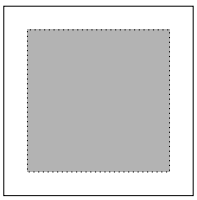


Figure 5.2: Valid object centers

$$moment1 = \frac{\sum_{i=1}^{n} f(x_i, y_i) * d(x_i, y_i)}{n}$$

$$moment2 = \frac{\sum_{i=1}^{n} (f(x_i, y_i) * d(x_i, y_i) - moment1)^2}{n}$$

These statistics were shown to be insufficient to solve the problem because of the localisation problem that is discussed further in Section 6.1. To address the localisation problem, the means of the four local regions shown in Figure 5.3 were included in the terminals. At the same time, all the other terminals except the mean of the whole window and the T value were removed. This then meant that the terminals were the four local region means, the mean of the whole input window and T. This was the final set of terminals used on the easy images.

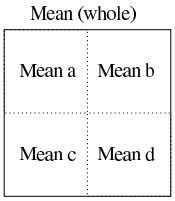


Figure 5.3: Local regions

5.2.3 Coins 5-10

To begin with for the coins images the statistics used were mean, standard deviation, first moment, second moment and T. Each of these statistics were calculated on the whole input window. These where calculated in the same way as mentioned for the easy images. Again these terminals were insufficient to solve the problem so local regions were introduced. The new terminal set created was the means of the four local regions, the mean of the whole input window and T. These terminals had the "flipped regions" problem that is mentioned in the results section of this paper. However over a large number of generations this problem was solved with these terminals. Due to the slow rate of training, the terminal set was extended to include the mean of the central region. This region was 1/2 the size of the original window and was centered at the input window as shown in figure 5.4. This was expected to solve the flipped regions problem.

5.2.4 Evaluating the Usefulness of Terminals

Initially the features that were extracted were the mean, standard deviation, first moment and second moment. Each feature was calculated based on the whole input window. Another

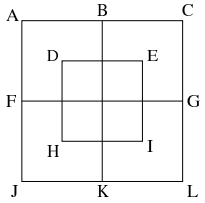


Figure 5.4: local regions

terminal was T. T is the user defined constant that is used for classification, as mentioned in Section 5.1. These terminals were used on the easy images and it was found that they were not expressive enough to satisfactorily detect the easy images. A more in depth look at why these terminals were not satisfactory is discussed in chapter 6 on page 22.

One of the benefits of using GP is that GP has the ability to select the terminals (and functions) that is "wants" from the terminal (or function) set. The use of the fitness function means that the programs that are created are measured on there accuracy of solving the problem. This measure is directly related to the terminals and functions in each program. Therefore if a program has a "good" fitness then the combination of terminals and functions in the program are "good". This means that the frequency of each terminal can be used as a measure of the relative importance of the each of the terminals. One thing to consider is the irrelevant terminals and functions, as defined in Section 3.7. Often programs can be reduced to half of there original size. This reduction is due to the irrelevant terminals and functions in the programs generated by the system. After the reduction is performed, analysis of the relative frequencies of the terminals can show the relative usefulness of those terminals.

It was clear from the results that when the detection was better there was a high number of means and T values were as the relative numbers of first and second moments were significantly less. As well as this the problem highlighted in the results section of edge detection of the class with the highest average pixel value meant that new features had to be used.

The new set of features contained just mean values and T. There were five means that were used. They were the mean of the whole and four local region means, as shown in figure 5.3 on page 18. These terminals were expressive enough to distinguish between squares, circles and background in the easy images. When these statistics were used on the coins, although the fitness was acceptable, the concern mentioned in results meant that one further statistic added to the existing set was the mean of a central region.

5.3 Functions

Functions are the operators that combine terminals to form programs that model the problem. This makes them a very important part of the genetic programming process. To begin with, the simple arithmetic operators were used, that is, $+, -, /, *$. The division operator was a protected division instead of the standard arithmetic division operator. The protected division operator returns 1 if any number is divided by zero. In addition, an if function was used. This function had three numeric arguments. The first argument was the condition, which is treated as true if the number is greater than zero, otherwise false. If the condition is true then the if function returns the second argument, otherwise it returns the third argument. These functions were found to be sufficient in these experiments. However some other functions were implemented (with limited success), including $Abs, Sine, Cosine, Exponential, Log, Min$ and Max.

5.4 Clustering

The task of object detection is a difficult one. It becomes very difficult to detect an object in the region centered at pixel (x,y) and at (x,y+1) detect background. This is because the two regions are so similar, as can be seen in Figure 5.5. Another problem is that during training if there is some small region in the image that is background but the program classifies it as an object of interest, the fitness function will count each of these pixels as false alarms. However, as all of the false alarms relate to one object it would be better to count it as a single false alarm. To solve these problems clustering can be applied to the outputs. The clustering algorithm used in this project has three steps:

1. Calculate the positions that objects are correctly detected and the number of times that the background is detected as an object of interest (this includes objects that are within tolerance of the true centers).

2. Count the number of true centers that have at least one detected center within tolerance of it.

3. Calculate the number of adjusted false alarms. Which is calculated by, for each center (x,y) that is in the list of false alarms, remove any other centers that are of the same class and that are between (x,y) and (x+tolerance,y+tolerance), and add one to the number of adjusted false alarms.

This then means that the number objects detected can be between 0 and the number of objects and the adjusted number of false alarms is the number of clusters that are incorrectly detected.

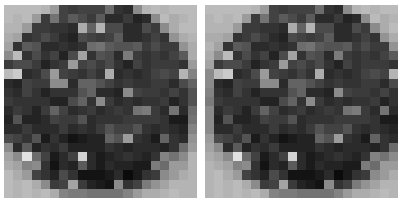


Figure 5.5: Example of difficulty of detection

5.5 Fitness

The fitness function is another very important part of the GP process. It is absolutely imperative to get the fitness function right. This is because the fitness function is the measure of a program's performance. The fitness function must allow a large change in performance to be shown with a large change in the fitness function. This is important for selecting programs for reproduction, crossover and mutation.

To begin with the fitness function was set to:

$$a * (1 - DR) + c * FAR$$

where a and c were constants that reflect the relative importance of the FAR and the DR. The DR and FAR are calculated by using the sliding window across the whole image. At each position of the sliding window, the output of the generated program is compared with the desired output. If the output does not match the desired output, then the number of false

alarms is incremented by one. If the output matches the desired output, then the number of objects detected is incremented by one. After the window has slid across the whole image(s), then the FAR is calculated by dividing the number of false alarms by the number of objects desired. The DR can calculated by dividing the number of objects detected by the number of objects desired. The fitness of the program can be calculated and be assigned to the program. This process occurs for every program in the population for every generation.

This process was found to have some limitations when clustering was used. Clustering was applied to group small regions that the program classified as the same type of object. This is because the program is likely to report that there is an object of some class not just at the true center but also at the pixels directly around the true center. These classifications can be grouped together and just the true center reported. As well as this case, the program could report that there are some objects of class 1 say where there isn't, if it then classifies all of its neighboring pixels as that class, then this should only be reported as one error instead of 9 (the true center and the eight directly around it).

As well as this problem there is another problem when clustering is used with FAR. Consider these two cases:

1. If there is a program that has a small cluster of incorrectly classified pixels, then the number of false alarms is one.

2. If there is another program that is identical to the first case except it has one less pixel in the cluster of incorrectly classified pixels.

We can see that case two performs slightly better than case one but their fitness will be the same. This could cause problems as the only way that the fitness can get better is by completely removing the whole cluster. This seems like an unreasonable thing to expect and it would be far better to allow gradual fitness improvement. For these reasons it was decided that FAR and DR were no longer sufficient to calculate fitness. We introduce False Alarm Area FAA to the fitness function, as shown below.

$$fitness = a * (1 - DR) + b * FAA + c * FAR$$

The FAA is the number of pixels that are False Alarms. That is the number of pixels without clustering that are incorrectly classified. Where again a, b and c are constant factors that reflect the relative importance of FAR,DR and FAA. We use this function throughout this project. The key to its success is that minimizing this function minimizes three things. The first is (1-DR), which is zero when DR is 100%, that is, all the objects were detected. The second is, the FAA is minimized when there are no pixels incorrectly classified, which is our goal. The last is the FAR, after considering what the minimizing the (1-DR) and FAA do, it would appear that FAR is not so important. While this is true, the FAR is a measure of the clustering of errors. This is particularly important if the system is not able to classify everything correctly. For example if say n pixels cannot be classified, it would be better to get $n/4$ clusters of pixels, than n individual pixels. This is because that in this case the output would need to be processed further, possibly by a person if this is the case n/4 clusters is far better than n pixels.

Chapter 6

Results

6.1 Easy

The easy task, which was detecting squares and circles was attempted in a number of different ways. The first was to use an input window with no local regions. Only pixels statistics of the whole input window were used. A problem this method had was where the input window contain part of a high pixel intensity object and part of a low pixel intensity object (or background). This situation occurs on the edges of the squares, in this case. The mean of an input window that contains half a square and half background is the same as the mean of a circle. This is because the square has a mean intensity of 200, background of 100 and circle of 150. This problem should have been overcome using the moment. However the use of the first and second moments where reduced to irrelevant positions in the programs produced. This means that the system was mainly focused on the mean statistic which is not expressive enough to distinguish between the three classes. The results of this approach were poor, with a detection rate of 100% but a FAA of 440 pixels. This FAA relates to the edge of the squares where there was a pixel line around the squares. Each square had approximately 40 pixels around it that were incorrectly classified. This problem clearly shows that only using pixel statistics on the total input window was not sufficient.

To solve the edge problem, local regions were introduced. These local regions are shown in Figure 5.3. The local regions can be used to asses the location of an object within the input window. This then gives the system enough information to solve the edge problem. The search space however has also been increased as another four terminals are added. These four terminals are the four local region means. To keep the search space as small as possible, the standard deviation, first moment and second moment terminals were removed. This means that the new terminal set contains the four local means, T (the Threshold) and the mean of the total input window. The results of these terminals being added was perfect detection of the squares and circles. One program that the GP system produced to solve the multi-class detection problem for the easy images is shown in Figure 6.1. An example of the detection outputs can be seen in Figure 6.2. This Figure shows the program in Figure 6.1 outputs. The top left image is the original image, the top right image is the image that relates to objects that are classified as background, the bottom left image is the detection of circles and the bottom right image is the detection of squares. In each image the the pixels are equal to the output within the range T. If the output is not in that class then the pixel is set to the background a pixel intensity of 80. If however the pixel has an output that is within the range of that class of object the pixel value is the output value minus, T times the $x - 1$, where x is the class number of the object. The images produced so the output values within the range of the T of that class of object.

**********8

Terminal	Pixel statistic
F1	Mean of whole input window
F2	Mean of top left region of the input window
F3	Mean of top right region of the input window
F4	Mean of bottom left region of the input window
F5	Mean of bottom right region of the input window

Table 6.1: Mapping of Terminals to Pixel Statistics

```
Blah
```

Figure 6.1: Program constructed to solve the detection problem for the easy images.

program goes here
****

The total training time however was still very slow. Initially the system quickly improved fitness however, after a number of generations the improvements of fitness slowed dramatically. As a result of this observation a new terminal was added. The new terminal was the central region as shown in Figure 5.4 on page 19. The result of adding this terminal was that the training time was slightly reduced.

6.2 Coins 5 - 10

The more difficult task, which was detecting 5 and 10 cent pieces on a uniform background was also attempted in a number of ways. To begin with the original approach used on the easy task was also used on the coins, the use of pixel statistics of the whole input window. This however had the same problem as the easy task, where the edge of one object was being detected as another class of object. Another problem with using only the pixel statistics on the whole input window is that, if an object is smaller than the input window. There can be multiple positions where the object is contained in the input window. This means that if the background is reasonably uniform then the mean of any input window that contains the whole image will be very similar. This is shown in Figure 6.3 where the central image in the figure is the true center but all three are very similar. This makes the problem of localisation on those pixel statistic impossible if it only uses the mean of the window. This means that the standard deviation, first and second moment need to be used to decipher between the cases where the object is contained within the input window. Unfortunately this did not appear to occur within a reasonable amount of training. As a result of these problems local regions where introduced. This solved both problems, however it did show another problem.

Local regions proved to be very important after the introduction of local regions the system was then able to localise objects more accurately. This approach was then able to produce perfect results of 100% detection and a FAA of 0. This was because not only was the system able to distinguish between half a 10 cent piece and a 5 cent piece but it was also able to distinguish between a 5 cent coin in the center of the input window and a 5 cent coin in the corner of the input window. The fitness reduced quickly at the beginning and then slowed down dramatically as it got close to zero. Looking at the programs when the reduction in fitness slowed dramatically it was observed that the system reported objects between two objects, as can be seen in Figure 6.4. These reported objects occurred in areas of background that had objects near it. On closer inspection of these locations however it was discovered that the problem was occurring when the input window spanned across two coins, as shown

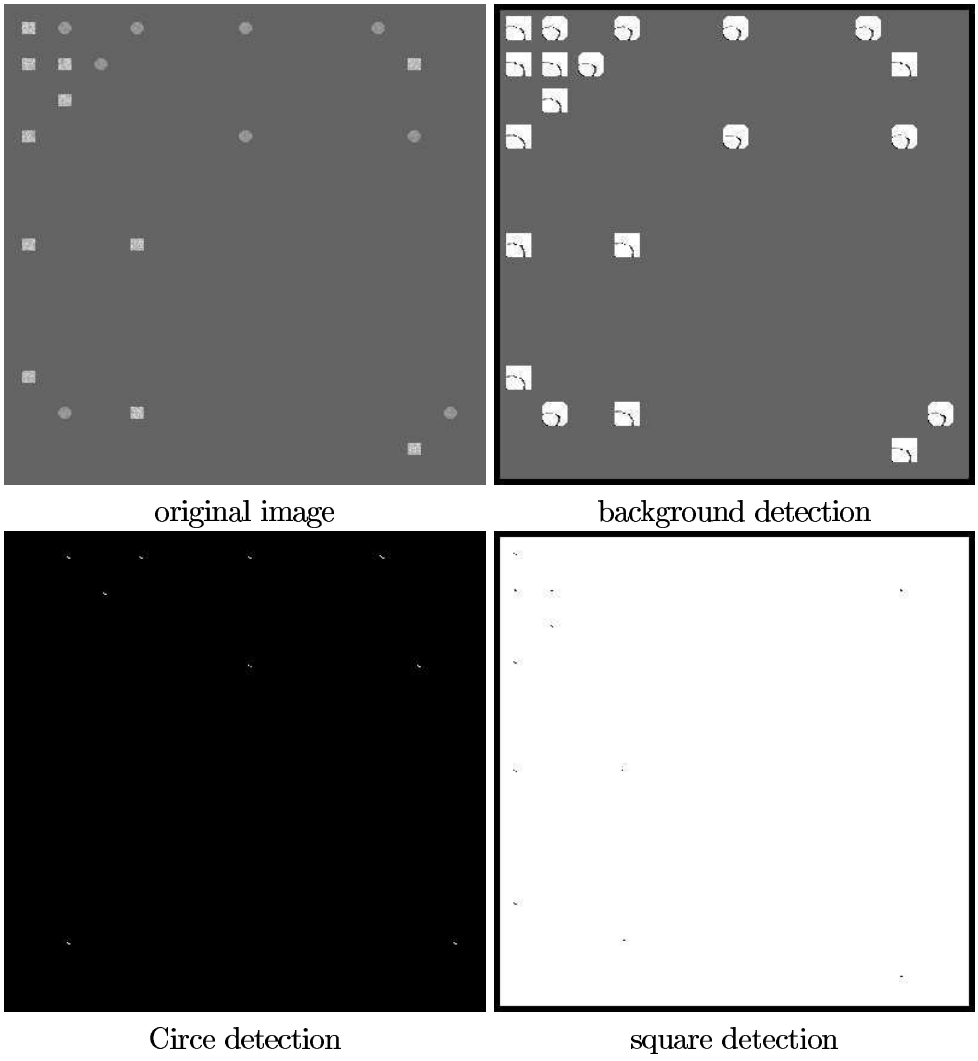


Figure 6.2: Detection results

in Figure 6.5. This problem clearly occurs because the system calculates means of the four local region and uses those means to evaluate if there is an object within that region. In the case shown in Figure 6.5 however there are in fact objects in each of the regions. This was because the input window can span across two objects. Despite this problem, as mentioned the system did in fact get perfect detection, but the training took 162 generations, with a population size of 500.

After observing the problem caused by the input window spanning two objects another mean of a local region was added to the terminal set, which is shown in Figure 5.4 on page 19. This new local region was the center of the input window. The region was made 1/4 of the size of the input window and centered at the input window. The case of the input window spanning two objects is distinguishable from the case where the object is in the center of the object. The result of this approach were again perfect, 100% detection and FAA of zero. This result came from a population of 500 after just 100 generations. The program that was generated is shown in Figure 6.6. The result can be seen in Figure 6.7. This shows the positions that the program suggested that an object existed. as can be seen the green pixels are at the center of each object showing correct detection of the objects. It is interesting to note that even though there was a tolerance level of two pixels, only 3 of the coins had

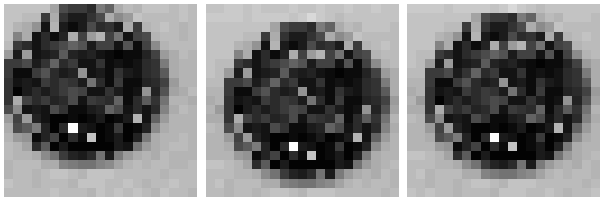


Figure 6.3: Example of difficulty of localisation

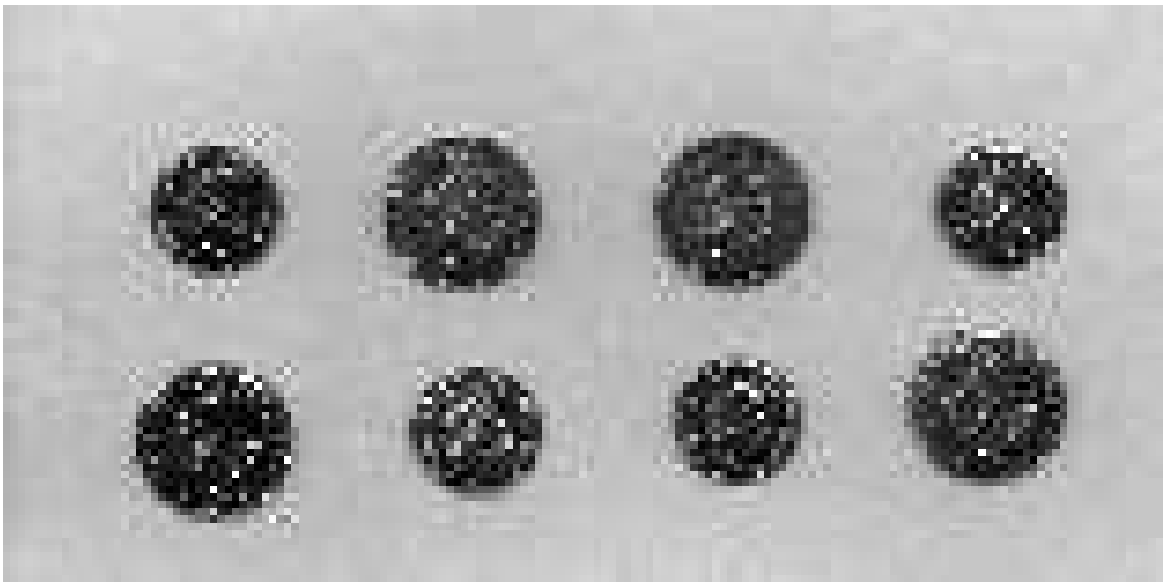


Figure 6.4: Output image showing result of the program shown in Figure 6.6, green pixels are correctly detected coin centers, red pixels are incorrectly detected coins

multiple detections within that tolerance.

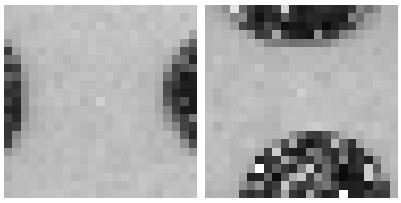


Figure 6.5: Example of problems with local region means

Terminal	Pixel statistic
F1	Mean of whole input window
F2	Mean of top left region of the input window
F3	Mean of top right region of the input window
F4	Mean of bottom left region of the input window
F5	Mean of bottom right region of the input window
F6	Mean of central region of the input window

Table 6.2: Mapping of Terminals to Pixel Statistics

```
(/ (* (/ (* (+ F4 (if (/ (/ (- (if (/ F5 F5) F5 F6) F3) F3) F5) T T)) (if (-
F2 T) F6 (if F6 (- F3 T) F4))) (- (- (+ F2 F6) (if F3 F6 F5)) (- (+ F1 F3)
(if F5 F 2 (- (if F1 (- F2 F1) F4) F2))))) T) (if (* (if F5 F2 T) (/ (if (*
(* (* T T) F4 ) F5) (* T F4) (if (* (if F5 F2 (- F2 T)) (* (- F3 T) (/ T (-
T F5)))) (+ (* (if F5 F6 F3) (if F1 F4 F3)) F5) F2)) (- (/ F2 F4) (/ F5 (if
T F3 (if (if (+ F2 F6) F6 F4) T F4)))))) (if (+ (if F3 F3 F6) (- (* T (if F6
F2 (+ F5 (- (+ F1 (+ F3 T )) (- (+ (/ F5 F2) F6) F4))))) F1)) (- F1 (if (*
(/ (* (+ (/ F5 F2) F6) (if (- F 2 T) (* T F3) (if F6 (- F3 T) F4))) (- (- (+
F2 F6) (if F3 F6 F5)) (- (+ F1 F3) (if F5 F2 (- F2 T))))) (/ (if (if (* (if
F5 F2 (- F2 T)) (* (- F3 T) (/ T (- T F 5)))) (+ (* (if F5 F6 F3) F5) F5)
F2) F6 F3) (- T F5))) F3 (/ F4 F2))) F5) (+ (* (- F6 F4) (if (- (- (+ F2 F6)
(if F3 F6 F5)) (- (+ F1 F3) (if F5 F2 T))) (if F3 F6 F5) (if F6 F2 F5))) (-
(* T (if (- (+ F1 (+ F3 T)) (- (+ (/ F5 F2) F6) F4)) F2 (+ F5 (- (+ F1 (+ F3
T)) (- (+ (/ F5 F2) F6) F4))))) (+ (+ (- F6 F4) (if F4 ( / F2 (+ (- F3 F2)
F1)) (if F6 F4 (* F6 F5)))) (* F5 F6))))))
```

Figure 6.6: This program is a solution to the 5 and 10 cent coin detection problem.

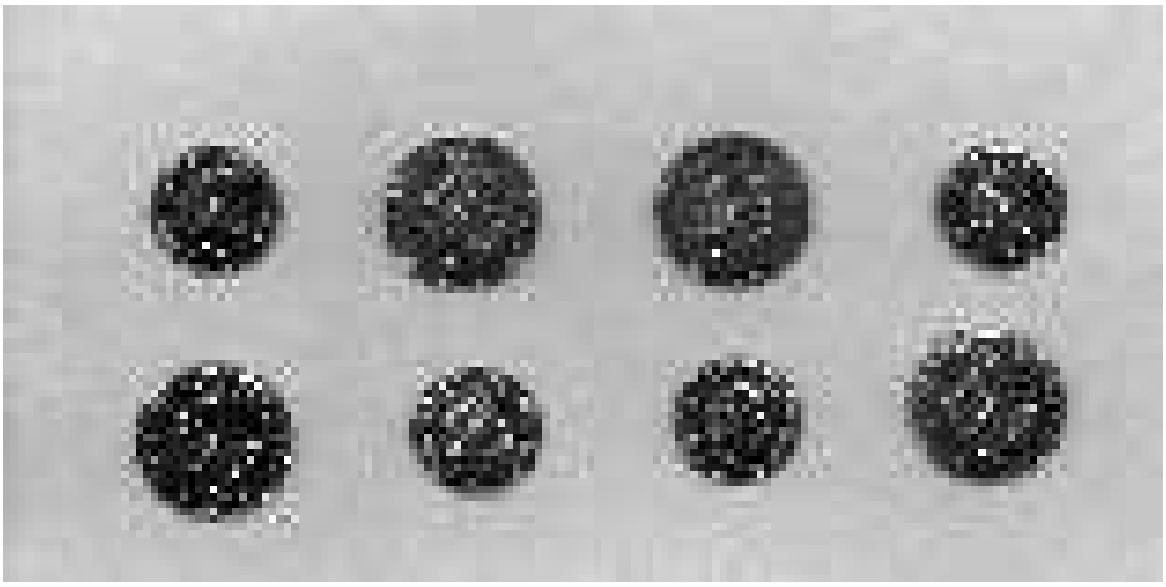


Figure 6.7: Output image showing result of the program shown in Figure 6.6, green pixels are correctly detected coin centers, red pixels are incorrectly detected coins

Chapter 7

Future Work

The results of this project showed that GP can be used successfully for object detection. However the approach is still not perfect. One of the main problems is long training times. Further work is needed so that training can be done faster. Particularly there are four things that have appeared from this project as possible solutions to this problem. They are:

1. Improving crossover and mutation.
2. Heating of the constants in the fitness function over time.
3. Dynamically allocating T values.
4. Sampling of fitness cases.

7.1 Improving Crossover and Mutation

There is some evidence that statistical analysis on nodes in each program can give a computed measure of the quality of a node. If this measure is found to be correct, then crossover and mutation can be applied at the worst point in the program. The measure however, is unlikely to be able to identify a single node of a program that is the worst although it is thought that the measure will at least be able to give a subtree and crossover or mutation can be applied to the subtree. The measure is simply the difference between the actual value at each node and the value that the node should have, assuming the rest of the tree is correct. The absolute value of this measure can be summed over the all of the fitness cases and the result can be used to find the worst nodes within a program. The standard deviation of the difference may also be a good measure of the nodes. This measurement should identify points in the program that are performing poorly within the program and they can then be removed from the program. If this measure is correct it will mean that crossover and mutation can be applied on the parts of the program that are performing the worst.

7.2 Heating of Constants

The fitness functions used in this project was:

$$(fitness = a * (1 - DR) + b * FAA + c * FAR)$$

This has three constant factors (a,b,c) that relate to the relative importance of the parts of the fitness function. At the begin of the GP process, the detection rate is very important as experiment results have shown that if objects are not detected early, they are not detected at

all. So constant a must remain high so that detection of objects is the most important part of the equation. The relationship between FAA and FAR is an important one as reducing one can increase the other. The FAA should be more important as the key goal is to get perfect detection with zero false alarms. FAA is more sensitive as a function as removing an incorrect detection point will always reduce the FAA. FAR however could report the same value when an incorrect pixel is added or removed. As a result, it is believed that FAA should, in the early stage, be higher than FAR. If the problem is not able to be solved perfectly however, FAR should be considered in the fitness function so that the pixels that are erroneously reported as objects are grouped together in a few small areas instead of single pixels scatter through out the image. The reason is that it then makes a manual check easier, as there are only a few places to look. It is because of this relationship that future research is suggested in the area of dynamically changing the constant factors. This would allow DR to be dominant to begin with and allow the FAA and FAR to change according to the stage of the GP system (i.e. number of generations and current fitness).

7.3 Dynamically allocating T values

The classification process can be considered a sorting process. The sorting only needs to place objects of each type within some range of outputs. These ranges can be anything as long as no two objects of class x are sorted where by there is another object of class y (where y is not equal to x) that is between them. This means that all objects of class x must be above or below all objects of y. In the experiments conducted for this report the classification process specified value ranges that each class of object must be within, which adds a further restriction to the problem. This restriction however is not specifically required for the objects to be classified. Further research in this area should look at the dynamic allocation of ranges. One possible strategy is to evaluate all of the fitness cases then set ranges for x say in the following way:

$$(min(x) + max(x-1))/2 \text{ and } (max(x) + min(x+1))/2$$

In this way the restriction of specific ranges is removed. The only restriction that this approach adds is that objects of class x must be above objects of class $x-1$ and below objects of class $x+1$. This restriction could also be removed but further research needs to be performed to examine whether the benefits of removing this restriction are worth while and exactly how to remove this restriction.

7.4 Sampling

The training process of a GP system on the image detection problem is very slow. As a result the use of sampling from the set of fitness cases would be worth investigating. Small images of 300 x 300 pixels contain 81,796 fitness cases (assuming a input window size of 14) and for hard problems multiple images may be required to train on it. This means that each program that is generated must be tested on thousands of test cases. This is highly inefficient as many of the evolved programs that are hopeless at solving the problem. It would be more efficient to begin by training on a sample of the fitness cases until the programs that are generated are more accurate then train on all of the fitness cases. It would even be worth training on the sampled data until the fitness was zero and then continue training from that point on with all of the test data.

7.5 Images

The experiments used in this project only contained two classes of objects of interest and the background. The approach was developed however, to apply to multiple classes of object. Due to time restrictions, these types of images have not been considered. However further research needs to be done to examine whether this approach can be used for detection problems with more classes

Further research is also need to examine what effect non-symmetric or non-circular objects have on the current process. The current techniques are effective for detection of circular objects but detecting other shapes may make the problem more difficult.

Another restriction on this research was that objects of all classes had to be the same size. Further research should examine the problem where objects of one class are much larger than the others. As well as this problem the case where objects of the same class can be variable sizes also needs to be examined. The use of size invariant pixel statistics is a starting point for this but there are a number of other questions that need to be answered. An example of the questions is, what size should the input window be?

The last aspect that needs further research is that of background. The backgrounds used in all of the experiments for this project were relatively uniform. This is an important future research area because a commercial implementation of this system is likely to have a cluttered background.

7.6 Terminals

There are two forms of terminals: 1)pixel statistics, and 2) common values. Common values are terminals such as T. These values should be limited as much as possible as they do not give the system useful information about the problem. Some however are useful, for example, T, which gives the system information about the classification strategy used by the system. This research did not examine possible common values extensively. The use of common values such as random numbers is one possible area of research. However a more useful aspect is examining pixel statistics. As there are a huge number of statistics that could be applied, it would be worth examining two questions. 1)Is there an ideal set of statistics for all detection problems? 2)Is there some way to examine a problem and predict accurately the pixels statistics that should be applied? The second question could be answered by calculating a large number of pixel statistics on a sample image and examining the relationship between outputs of objects of the same class and the relationship of outputs of all objects in different different classes.

Chapter 8

Conclusion

The goal of this project was to develop a domain independent approach to multi-class object detection problems using genetic programming techniques. Three terminal sets based on domain independent, low level pixel statistics were developed. The function set was based on the four arithmetic operations and a newly defined conditional function. The fitness function was developed based on the detection rate, false alarm rate and the false alarm area. The goal was achieved on the easy detection and the coin detection problems.

During the development and experiments of this project the following characteristics can be concluded as important to successfully solve the multi-class object detection problem with GP:

- Firstly, the results showed that for object detection an important characteristic is that the inputs include pixel statistics on local regions not just on the whole input window. These local regions should be in the form of Figure 5.4 on page 19. These local regions are important for localisation of objects in a multi-class object image.

  Figure 6.3 on page 25 shows the problem with only pixel statistics on the whole input window. The first problem is that there could be an object contained within the window and as long as that object is fully contained then the pixel statistics will be the same when the object is in the center of the input window and when it isn't. The second problem of the multi-class object detection also makes it difficult when part of an object is contained in the window and the other half of the window is background. In this case, a statistic like the mean of the while input window is likely to classify this situation as an object since the mixture of background and object become similar to the mean of an object. Local regions can solve these problems by giving the GP system more information about the localisation of objects within the input window.

- The second important characteristic is a function set that includes a conditional function. This is important as it gives the GP system the ability to model non-linear relations.

  From a theoretical point of view, a conditional function such as an `if` allows the GP system to build programs in the form of:

  $$(if,\ \text{object is class 1, then output } x, if, \text{object is of class2, then output } y \text{ etc})$$

  Given a situation where class one objects had a mean value of 100, class two had a mean of 230 and class three a mean of 150, there is no need to change the order in which the classes are arranged. If there was no `if` function, then this problem would be impossible to solve. The problem is not linearly separable and thus the system will

fail to solve the problem accurately. However if the classes were linearly separable, then the conditional statement is not as important. For example, the classes were ordered such that class one is the objects with mean values of 1, class two were the objects with mean 150, and class three were the objects with mean 230, the problem becomes linearly separable.

- The third characteristic is a good fitness function. As the value of the fitness function is what is being minimized, it is important that the fitness function is an accurate measure of the effectiveness of a program to model the solution.

  In the case of object detection, it is crucial that the fitness function contains two things. The first is some measure of the false alarms; the second is a measure of the number of objects detected. The two fitness functions that were discovered to be effective for multi-class object detection with clustering of classification were:

$$Fitness = a * FAA + b * (1 - DT)$$

$$Fitness = a * FAA + b * (1 - DT) + c * FAR$$

  For easier problems where the evolved program is expected to have 100% DT and 0 FAA, the first fitness function should be used. This is because the function is simpler and there is no way that FAR and FAA can cause problems as FAR is not contained in the function. However for harder problems where the output of the GP system is not expected to produce a program with 100% DT and 0 FAA, then the second function should be used. This puts some importance on the grouping of the incorrectly classified objects thus reducing the number of positions that need to be manually examined.

It is interesting to note that contrary to [5, 4] the parameters that control the GP algorithm did not appeared to be critical to the success of the system. The only two parameters that did affect the success of the system were that of the population size and number of training generations. This indicates that although the parameters may affect the performance of the system, the actual values of the parameters were not as important as the 3 key characteristics mentioned above. This means that when using GP for object detection, most attention should be paid to the 3 key characteristics rather than the parameters that the GP system uses.

Although the results are good, there are limitations to the use of GP for object detection. Some of those limitations are:

- All objects need to be similar sizes and objects of the same class must be the same size.
- Training is very slow. This means that the solution could take a number of days to be trained.
- The use of these techniques on mor difficult images.

Appendix A

Object Oriented Genetic Programming Package

This package was written by Peter Wilson in 1998. It is a package that was written for use in research. The package is a strongly typed genetic programming package. This means that the package is capable of using functions and terminals that are of different types. This means that rather than just using floating point numbers as this project did the package will support any other type. The package was initially written for the regression problem.

To convert the package from the regression problem to the multi-class object detection problem, a number of changes need to be made. They were:

- A way to read and store feature vectors from a file.
- A new fitness function.
- A classification strategy.
- New terminals for each pixel statistic used.
- A way to visualise outputs of the system during training to provide more information about the current state of training.

As well as implementing these new things to allow the system to work for the multi-class object detection problem, a number of bugs in the existing package were found. These bugs were difficult to find and a large amount of time was spent locating and removing them.

Bibliography

[1] S. C. Roberts D. Howard and R. Brankin. *Target detection in SAR imagery by genetic programming.*, chapter 30, pages 303–311. 1999.

[2] Mattias Fuchs. Crossover versus mutation: An empirical and theoretical case study. Technical report, Australian National University, 1997.

[3] Chris Gathercole and Peter Ross. Small populations over many generations can beat large populations over few generations in genetic programming. Technical report, Department of Artificial Intelligence, University of Edinburgh, 1997.

[4] John R. Koza. *Genetic Programming on the programming of computers by means of natural selection.* MIT Press, Cambridge, Massachusetts, 1992.

[5] John R. Koza. *Genetic Programming on the programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, Massachusetts, 1994.

[6] David J. Montana. Strongly typed genetic programming. Technical report, Bolt Beranek and Newman, Inc., 1994.

[7] W. A. Tackett. Genetic programming for feature discovery and image discrimination. pages 303–309. University of Illinois at Urbana-Champain, 1993.

[8] Jay F. Winkeler and B. S. Manjunath. Genetic programming for object detection. 1997.

[9] Mengjie Zhang. *A Domain Independent Approach to 2D Object Detection Based on the Neural and Genetic Paradigms.* PhD thesis, Department of Computer Science, RMIT University, Melbourne, 2000.