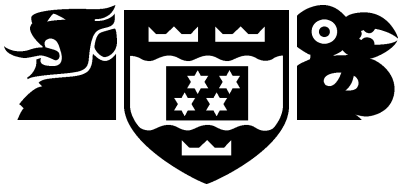
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



Computer Science

PO Box 600
Wellington
New Zealand

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

A Domain Independent Approach to Multi-class Object Detection using Genetic Programming

Urvesh Bhowan

Supervisor: Dr Mengjie Zhang

October 24, 2003

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

Abstract

Object detection is the process of automatically finding objects of interest within an image. Given the abundance of information now captured and stored in electronic form, this is fast becoming a useful and challenging machine learning and computer vision task. This project uses a domain independent genetic programming approach to solve four multi-class object detection problems ranging in difficulty levels. Three methods were investigated, all using different features sets based on local-region pixel statistics. A new measure, *program size*, was also introduced to the fitness function with the aim of favoring the evolution of smaller programs over larger, complicated ones. The new fitness function with program size proved more effective and efficient, and the evolved programs using this fitness function were much shorter and easier to interpret. While the classification method greatly reduced training times than the basic detection method, it could not improve the detection performance. The two-phase method with a secondary training phase always gave a better detection performance than without.

Acknowledgements

I would like to thank my supervisor Dr Mengjie Zhang for his support and useful feedback throughout the duration of this project. I would also like to thank the honours students in *Memphis* namely, Ben, Bunna, Simon, Jerome, Justin, Annie, Donald, Will and Richard, who were always working hard but never too busy to lend a helping hand, and Jane, for putting up with me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As more and more images are captured and stored in electronic form the need for programs to *automatically* find objects of interest in a database is increasing. In many cases, it takes highly trained experts in the field to find and classify these objects, but there is either too much data for the limited number of such experts or the cost of these people is too high. Creating automated systems robust and accurate enough to apply to the very wide range of fields and applications object detection covers is a difficult task.

Images can contain single or multiple classes of objects-of-interest, where these objects could be the only objects present in the image or they could be interspersed with other less interesting objects also present in the image. Furthermore, the range of backgrounds in such images also varies from being simple and uniform to highly cluttered and complex.

An application for object detection systems currently being developed is automatic target recognition (ATR) systems [15]. For example, accurate and automated ways of finding and locating target objects from images collected by ground or air surveillance units are required, where targets may range from helicopters, tanks or buildings. Weather monitoring and prediction systems are another area where automated object detection may prove useful [5]. Consider a warning system which routinely scans multiple satellite-generated weather maps to find and locate all occurrences of unusual cloud patterns having the potential to develop into hurricanes or tornados (figure 1.1). Apart from classifying whether a cloud pattern is unusual, dangerous or normal, there is also a strict requirement in reporting exactly *where* these occur.

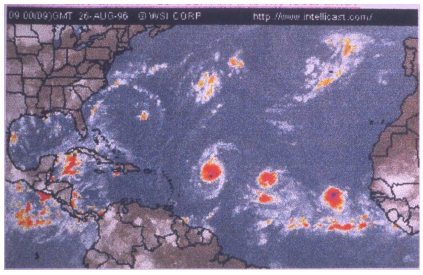


Figure 1.1: A satellite-generated image showing many possible hurricanes

In both applications, the object detection system is responsible for two tasks: accurately finding all the objects of interest in the image and categorizing them into different classes of object of interest.

For robustness, we also require an approach to be *domain independent* in this project. That means that the same method of object detection should work unchanged on a range of dif-

ferent problems and applications. By enforcing this constraint, no prior domain knowledge about the objects of interest such as the shape, size and orientation, is required to perform the detection task.

Genetic programming is a useful machine-learning and search technique which can provide a general solution to a task-specific problem such as object detection [12]. It is the *automatic programming* or *evolution* of computer programs to solve a problem, where such programs learn about their environment through the context of a task.

Genetic programming (GP) is based on the principals of Darwinian evolution or natural selection applied to computer programs. A program or individual in a population is assigned a fitness depending on their ability to solve the problem. The programs with the best fitness are then selected for recombination resulting in the next generation of programs being fitter (or at least not worse) than the original. This process terminates when an optimal solution is found or when the best individual can no longer improve over a number of generations.

Since the late 1990's, GP has been applied to object detection problems. Most work done in this area either focuses on binary object detection problems, or a multiple stage approach involving extensive 'processing' of the image data. The former involves simplifying the problem down into only two classes: an object of interest and everything else (background) [19]. The main disadvantage with this approach is that it does not consider problem involving multiple classes of interest, as is the case with most real-world data. The latter involves applying multiple domain dependent image-processing techniques on the raw images such as edge detection and segmentation before the actual detection/classification step in the goal of making this most important final step simpler [9]. The main disadvantages with this approach are that the image processing techniques often require prior knowledge about the problem domain and the final results are often too heavily influenced by the degree of success of each of the previous (image processing) stages.

This project uses image features based solely on the raw pixel values to ensure domain independence. The feature extraction involves using pixel statistics on different groups or regions of pixels to obtain higher level features about the correlations of the raw pixels in the image data.

1.1 Goals

This project aims to develop a domain independent approach to object detection using genetic programming with the goal of achieving efficient training, good detection performances and comprehensive genetic program detectors. Four object detection tasks of increasing difficulty will be used for the experiments. Specifically, this project aims to address the following research aspects:

1. From the three object detection approaches developed in this project, namely straightforward detection (SFD), object classification applied to detection (OCAD) and refining object classification for detection (ROCD), the following research questions are investigated:

    - Can the SFD method achieve a good performance on the four detection tasks?
    - Can OCAD method more efficiently train good program solutions to achieve good detection performances?
    - Can the ROCD method improve the object detection performance over the OCAD method?

- Which of these three methods will perform the best on the four detection tasks?

2. To achieve the goal of domain independence, five terminal sets based on pixel statistics will be extracted from local regions in the images. Specifically, the following questions will be investigated:
   - Are circular region features more effective than square region features on these object detection tasks?
   - In terms of recti-linnear features, is the simplist feature set, a single central quadrant, powerful enough for these detection tasks?
   - Which terminal set can perform the best on the four tasks?
3. The *program size* will be introduced to a commonly used fitness function for object detection to form a new multiple-objective fitness function. Specifically, this project will investigate whether the genetic programs evolved using this new fitness function are shorter and more comprehensive than those programs evolved without using the program size in the fitness function.
4. This project will also address whether the object detection performance will deteriorate as the degree of difficulty of the detection tasks increases.

1.2 Contributions

This project shows how to both evolve genetic program detectors using object cutouts, and refine these genetic program detectors using a secondary learning phase based on the window-sweeping method on the entire images in the training set.

This project shows also how to evolve more comprehensive genetic programs through the use of a multi-objective fitness function. The *program size* was added as an additional contraint to the fitness function to explicitly favour the evolution of smaller programs. Part of this work has been submitted to the first *Australasian Workshop on Data Mining* for review and publication: `Program Size and Pixel Statistics in Genetic Programming for Object Detection`

1.3 Structure

The rest of this project is split into five chapters. These are organized as follows. A brief literature review is presented for both genetic programming and object detection in chapter 2. The list of the object detection tasks/problems in chapter 3. The methodology used in approaching these tasks in chapter 4. The various experiment results in chapter 5, and the conclusions and future work in chapter 6.

Chapter 2

Background and Literature Review

This chapter is split into five sections. The first and second outline a brief overview into machine learning and genetic paradigms respectively, followed by a more detailed discussion about genetic programming, important computer vision and image analysis tasks, and finally a discussion into genetic programming for object detection.

2.1 Overview of Machine Learning

Machine learning is a broad and rapidly developing area of research. Different artificial intelligence experts in this field vary in their definitions of what exactly constitutes machine learning but most agree the central idea involved computer programs which learn to solve problems without explicitly being programmed or told how to do so [2][4][12]. There are also many different learning paradigms which aim to address this issue, some of which include [16]:

1. *Connectionist paradigms* such as artificial neural networks,
2. *Genetic paradigms* such as genetic algorithms and the more recent genetic programming,
3. *Case-based* or *instance based learning* paradigms such as nearest neighbor or nearest centroid,
4. *Induction-based learning paradigms* such as decision trees or condition-action structures.

A learning procedure is also extremely difficult to define. Traditionally, learning has been split into three fields or strategies: supervised, unsupervised and hybrid [2][11].

Supervised learning is learning with a teacher. In the context of machine learning, it is where the desired output or answers to a problem are known and the learning system tries to find rules to produce its output or answers as close to the correct ones as possible. Common to supervised learning problems are the concepts of *training* and *test* sets. A training set is a collection of input patterns from which rules or features are induced. A test set is a similar collection of input patterns, except that these are not used and remain unseen while learning the rules. The purpose of the test set is to evaluate the performance of the learnt rules on unseen instances of the problem. In such learning systems, the procedure for learning then becomes two-fold: to discover or learn some kind of knowledge or rules from a problem or training set, and apply these rules to the test set to measure how well the learnt concepts perform on unseen (future) instances of the problem [16].

Unsupervised learning is learning without a teacher. That means, there is no correct answers for the learner to explicitly learn from. Instead it must explore underlying structures

or correlations present in the data to learn relationships rather than rules. Unsupervised learning most often involves learning knowledge by associating or summarizing new information with previously known knowledge [2].

Hybrid learning combines supervised and unsupervised learning, where parts of the overall learning are done through supervised while others through unsupervised learning.

2.2 Genetic Paradigms

Genetic algorithms (GA) is a machine learning or search technique based on the principles of modern biological evolution, inspired by ideas from the publication of Darwins revolutionary theories about the evolution and natural selection of species [2][12]. Those theories assert that only the fitter organisms in an ever-changing environment survive and reproduce while the less fit merely die off. The offspring of those fit organisms will then have similar or the same genetic code as their fit parents thus resulting in the new generation of organisms being fitter or at least as fit as the current generation.

In the 1960's, Holland adapted this model to apply to artificial systems [16]. It involved generating a population of individuals, where each individual represents a potential solution to a given problem. Individuals or *chromosones*, are typically encoded as fixed-length bit strings (usually binary), where each element in this string is called a *gene*. An optimal solution will be made up of the optimal genetic code, having all the 'best' genes in its chromosome, and will be relatively fit. An individual's fitness is measured against how well it solves the given problem or task. Members of the population are then selected for recombination depending on their fitness and a new generation of potential solutions is generated. This new generation should be just as fit or fitter than the previous generation.

The two main genetic operators used in GA are *crossover* and *mutation*. Crossover replaces the genes of the different chromosones at randomly selected points to form two new offspring, and mutation replaces a randomly selected gene in a chromosome with another. This evolutionary process is continued until an individual has reached the optimal fitness or a fixed number of generations are reached. The goal like natural selection, is to have some useful part of an individuals genetic code propagated down generations until the optimal individual is evolved [12].

One major problem using a GA approach is that the genetic algorithms often require a complex encoding/decoding process. The encoding process is required to transform a task into the appropriate *chromosone* representation for the evolutionary process. The decoding process is required at the final stage of evolution to convert the individual (solution) from the *chromosone* representation back into a more useful solution representation. Consider the chromosone presented below in figure 2.0. Without the specific encoding/decoding knowledge, its meaning is almost impossible to interpret.

1	1	0	0	1	0	0	1

Figure 2.0 Example of an encoded 8 bit binary chromosome (individual)

2.3 Overview of Genetic Programming

Genetic programming (GP), like GA is a kind of machine learning and search technique. It extends the idea of genetic algorithms by increasing the complexity of problem representation from fixed length bit strings to *computer programs* [12]. These computer programs are typically represented as tree-like structures which can vary in length as well as what input alphabet they may contain.

A program, like using GA, is assigned a fitness depending on its degree of success in achieving some task. This is determined using a fitness function. A population of genetic programs is then mated using the crossover, mutation and reproduction genetic operators in the goal of eventually finding the optimal program (solution). Crossover mates 2 individuals by randomly swapping their sub-trees, mutation randomly replaces an individuals sub-trees with another and reproduction copies an individual straight into the next generation without altering their genetic makeup.

The evolutionary process works on the idea of building blocks using program sub-trees [3]. Good (fit) programs will survive, preserving their useful subtrees or attributes. When mated with other programs, these useful attributes will be spread and combined with the genetic makeup of other individuals thus increasing their fitness.

GP can also be thought of as a beam search technique, where a program represent a possible solution in the *solution-space* [12]. This search is done in parallel as each individual can be seen as a separate search. No attempt however is made to learn the relationship between good individuals and the environment as the goal of evolutionary programming involves simply finding one individual that maximizes a given fitness function.

2.3.1 Program Representation

Programs in genetic programming are represented as a variable-size tree of expressions. The leaf nodes correspond to *terminals* and the non-leaf nodes *functions*. Much GP work was done using LISP or LISP-like representation of programs [12]. The algebraic equation $(x^3 + 2x)$ can be therefore be represented as a LISP S-expression `(+ (* x (* x x)) (* 2 x))` where the arithmetic operators form the functions and the variables (`x` and 2) the terminals. The corresponding tree structure is presented below in figure 2.1. The square nodes correspond to functions and the oval nodes the terminals (leaf nodes).

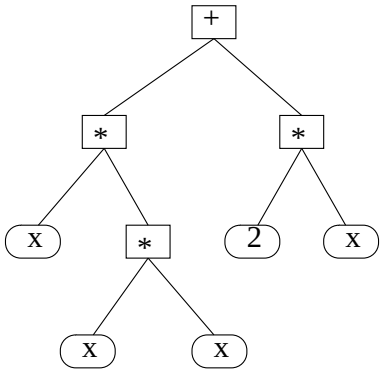


Figure 2.1: Program representation in GP

Functions and Terminals

The alphabet accepted by genetic program expressions are constructed using a function set and a terminal set [12]. Unlike GA these can represent a very large alphabet, and are usually user defined to fit the context of a given problem.

The functions form the root and internal nodes of the program. They may include arithmetic operations $\{*, -, +, /\}$, mathematic functions such as {sin, cos, exp, log}, boolean operations {AND, OR, NOT} or task specific user-defined operations such as {`move_left`,

`move_right, move_back`}. A function can contain other functions or terminals, as long as the function takes the correct number of arguments.

Terminals form the leaves of the programs and therefore take no arguments. They are usually some data types consisting of atoms representing possible inputs, sensors, variables in some system or constants.

The selection of function and terminal sets are crucial to the success of solving a program [16]. Both function and terminal sets must satisfy the requirements of closure and sufficiency [12]. That is, a function should only be able to accept as its arguments, any value or data type that may possibly be returned by any another function, and any value and data type that is defined in the terminal set. A bad (incomplete or inappropriate) selection could result in the slow convergence of a solution or the evolutionary engine not even being able to find the optimal solution at all.

Fitness Function

The fitness function is very important to genetic programming as it is the only measure of how an individual has performed on the task [16]. A fitness function is designed to grade a programs performance on the training set and provide continuous feedback on improvements or areas of difficulty [3].

2.3.2 Operators

The genetic operators *crossover, mutation* and *reproduction* are the essence of the evolutionary engine. It is where the individuals are genetically altered depending on their fitness. The number of individuals on which to perform each of the three operators is defined by user controlled parameters namely *crossover rate, mutation rate* and *reproduction rate*. This parameter is usually in the form of a percentage of how many individuals to apply each of the three operators to. Each operator is briefly described below.

Cross-over

Crossover is arguably the most predominant operator used in GP [12]. It is performed on two individuals from the population, producing two offspring. It works by selecting two individuals from the population, and swapping a randomly selected subtree from one individual with another randomly selected subtree from the other individual. The subtrees selected for crossover are chosen from two randomly crossover points from both individuals. Crossover works on the idea of combining certain attributes between two individuals.

A strong implication of crossover is that programs with a better fitness are crossed over more than those of weaker fitness. This continues the idea of 'survival of the fittest' as the next generation becomes influenced by the stronger genetic makeup (subtrees) of the fittest individuals from the current generation. The ideal case is if programs A and B are chosen for crossover where A and B have partially correct trees, and the crossover points occur precisely on these partially correct subtrees, then the result is two offspring with fully correct trees [14].

This however is rare as isolating the right 'partially correct subtree' is difficult when selecting the crossover points. Often the crossover points end up disrupting or breaking-up a partially correct subtree, replacing it with a less useful one. This problem is thought to occur more often in the later stages of the evolutionary process when the partially correct subtrees are larger and more difficult to isolate than in earlier stages.

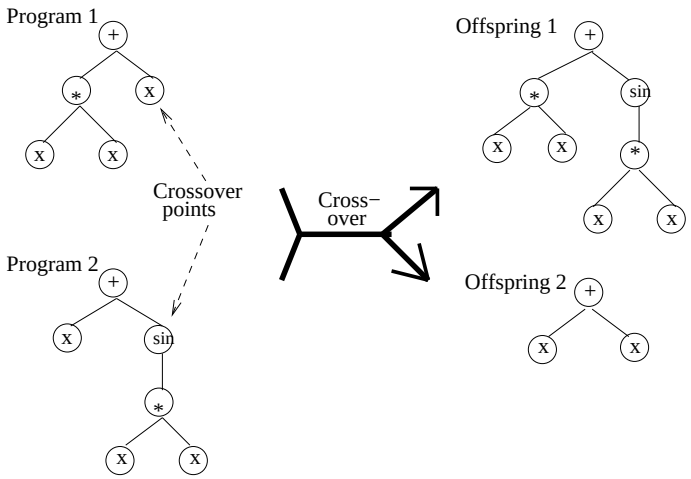


Figure 2.2: Crossover operator in GP

Mutation

The mutation operator is the only one of the three operators to introduce new genetic code into a population via an individual during evolution [16]. It is performed on one individual from the population, producing one 'offspring' which is the mutated version of the parent. Mutation works by randomly selecting a subtree from the individual chosen for mutation, and replacing it with another randomly generated subtree. This new subtree is generated in the same way the initial population is generated (via the grow, full or ramped half-and-half method).

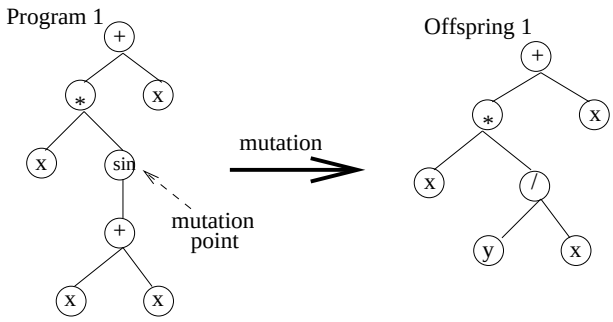


Figure 2.3: Mutation operator in GP

The motivation for mutation is to keep up the diversity of the population [12]. The ideal case here is if program P1 is *almost correct* except for one subtree B, and if B gets selected for mutation and is subsequently replaced with a more useful subtree C, then program P1 becomes optimal [14]. But like crossover, selecting the right mutation point is crucial. The worst case is if program P2 is almost correct, and a mutation point is selected which replaces a useful subtree in P2 with one less useful, the resulting mutation will be *worse* than the original parent.

Reproduction

Reproduction is the simplest of these genetic operators [16]. It involves simply copying a selected individual from the current population into the next population. By not altering any of the genetic makeup of the individual chosen for this operator, unlike crossover and mutation, the resulting program in the next generation can not ever be worse (less fit) than the original. For this reason, only the top programs from a population are selected for reproduction. This is important as it ensures that the evolutionary process produces individuals that can not possibly be worse (less fit) than the previous generation.

2.3.3 Evolutionary Engine

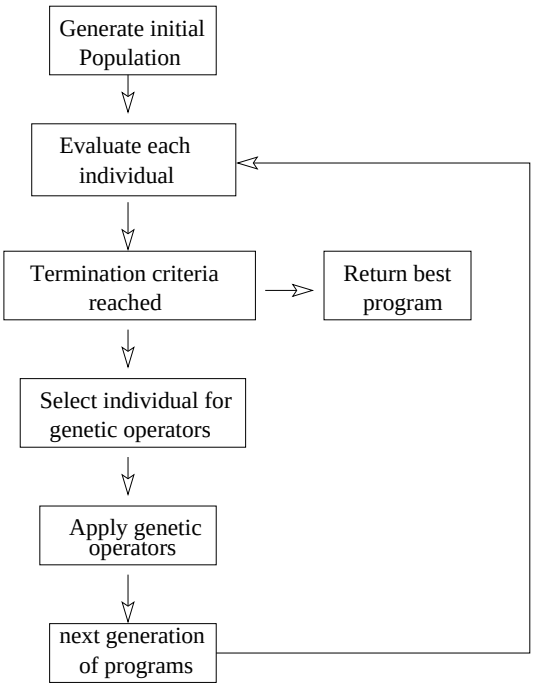


Figure 2.4: Evolutionary engine in GP

The initial population is randomly generated according to some user-defined parameters supplied to in the evolutionary process, usually the *population size*, the *maximum program depth* a program can grow to, the *minimum program depth* allowed for all programs and a *maximum number of generations*. There are several ways of generating these random programs to make up the initial population: *full, grow* and *ramped half-and-half* [12]. The full method ensures that full, entirely balanced program trees are constructed. This is achieved by selecting inner or parent nodes only from the set of functions and leaf nodes only from the set of terminals. The grow method allows different length subtrees in the program to be constructed by selecting intermediate parent nodes from either the function or terminal sets. This results in part of a tree terminating whenever a terminal is selected for that node as it becomes a leaf node. Ramped half-and-half combines both program generation methods.

Once the initial population is constructed, the fitness function is applied to each individual. After all the programs have been assigned a fitness, each is checked against the stopping criteria to determine if a program with the optimal fitness has been found. If so, the evolution terminates and reports the fittest program as the solution. If not, the population is

evolved using the three genetic operators, crossover, mutation and reproduction, resulting in a new generation of evolved individuals. This process continues until the optimal solution is found or the number of generations reaches a certain user-controlled number.

2.3.4 Current Issues in Genetic Programming

There are a number of issues that are currently being researched in this area and needs to be carefully considered when using GP. Some such issues are presented below.

- **Long training times:**
  Long training times are due to all the individuals in the population being evaluated against the entire training set to obtain a fitness, on every generation. The maximum number of generations allowed can range between 50 for easier problems or 200 or more for difficult problems.

- **More powerful representation of genetic programs:**
  Currently, genetic programs are represented as tree-like structures, but this is certainly not the only possible representation. There is research into other representations such as vectors, decision lists and even as procedural programs [7] (where procedural closely resembles a high-level programming language)

- **Improving GP Crossover:**
  Although crossover is a useful genetic operator in program evolution, there is much debate about *how* useful it actually is. This is because the crossover points are selected randomly, paying no consideration to the importance of separate subtrees to an individual program. Researchers are looking for ways to improve crossover by a more intelligent and effective selection of the crossover points. The issue then arises of how to separate and evaluate the usefulness of sub-trees within an individual without disrupting it.

- **Crossover** *vs* **mutation:**
  There is a considerable debate in this research area on which of the operators between crossover and mutation, is the better for the evolutionary process, and which should be utilized more in order to achieve the best results.

- **Overtraining:**
  Overtraining, or overfitting, is a problem inherent to such supervised learning paradigms. It occurs when the learning system learns the training data very well but not the test data. This implies that the learnt knowledge was not generalized enough as the underlying cause or model behind the data was not captured. Although this problem has been partially theoretically solved through the use of a *validation set*, over-fitting can still be a problem, especially if the model being learnt is very difficult to capture.

2.4 Computer Vision and Image Analysis

Computer imaging is the 'acquisition and processing of visual information by a computer' [1]. This spans a broad field, but can be split into two main categories: *computer vision* and *image processing* [16]. The former is the processing of images for use by a computer, and latter is the processing of images for use by people. This investigation focuses on aspects concerned with the former – computer vision applications – where images are examined and acted upon by a computer. Some real-world applications of computer vision include:

1. *Law enforcement and security*: scanning, processing and searching fingerprints though a police database; DNA analysis.

2. *Weather prediction*: scanning satellite generated images for warnings of interesting or unusual weather patterns such as cyclones or tornados.

3. *Medical applications*: scanning x-ray images for bone tumour diagnosis, retinal blood hemorrhages or unusual artery damage.

4. *Object recognition*: automatic target recognition, vehicle tracking systems or face detection.

5. *Robotics*: improving signal processing or enhancing robot-vision.

Computer vision tasks usually require some sort of *image analysis* be performed on images to solve problems. The process of extracting useful information from such images is called feature extraction, where useful information usually corresponds to image features. Image features can be grouped into three main levels:

- **Raw image intensities or pixels** (low level): These are domain independent and easy to extract, but a major limitation is that processing raw pixels is computationally expensive especially since much image analysis is done on large, complex and highly cluttered images such as satellite-generated images [15].

- **Pixel statistics of groups or regions such as mean, variance and first-order moments** (low level): These are also both domain independent and easy to extract, as the same statistics can be used on a range of different images containing a range of different types of objects of interest. Pixel statistics also offer an additional advantage over raw pixels – they can often capture relationships or correlations between neighboring or grouped pixels, something not immediately apparent from observing each pixel as an unrelated value [18].

- **Relational features** (high level): Relational features are highly specialized and domain dependent, usually requiring a field expert with good prior knowledge about the images domain to define. A limitation is the cost, effort and domain dependence attached to such a specialized approach to feature extraction.

2.4.1 Pixel Statistics For Feature Extraction

Feature extraction is the process of representing features extracted from an image as numeric-based feature vectors [14]. These feature vectors are often then used as input to image analysis tasks [18].

This project uses low level pixel statistics as features calculated on regions of pixels contained within an *input window*. The selection of what such regions or groupings of pixels is a difficult question. This is an important and active area of research as the choice of regions will impact on the 'quality' of the features extracted from the input window [14]. Quality here denotes how accurately the pixel statistical features capture or reflect the underlying raw pixel values contained within a given region. For example, the input window can be split into circular or square shaped regions which can in turn be split into smaller regions to capture the most amount of high level information as possible. Figure 2.5 illustrates two sets of regions, each split into different sizes.

Feature extraction is also considered a reliable and efficient *dimensionality reduction technique* [13]. That means, it can be used for extracting a new and smaller set of data (image

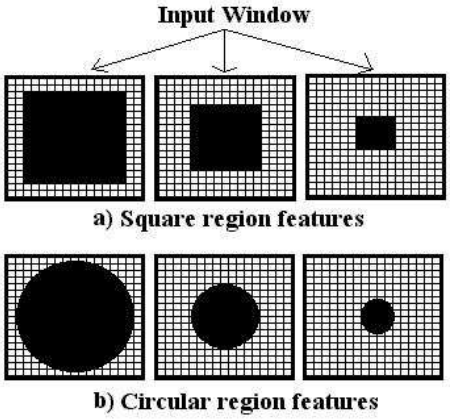


Figure 2.5: Possible regions for feature extraction

features like pixel statistics) from the original set (raw pixels), while keeping as much of the information from the original set as possible. This reflects the advantage of using image features as opposed to the raw pixels.

2.4.2 Object Classification and Machine Learning

Classification is the method whereby inputs are mapped to given output classes or categories. Object classification involves applying the same idea to images, where the input is an image or *object-cutout* and the output is the class which that image or object-cutout belongs to. Object classification is often the most important, final step in image analysis [16].

An object-cutout is a sub-portion of the original image contained within a given input-window. The input window is large enough to fit a single non-overlapping object of interest from an image. The goal of a classifier is then to correctly classify those image cut-outs into their correct classes.

Most classifiers use a machine-learning approach to find rules to correctly categorize inputs to output classes. This is done using the concepts of a training and test set [16]. The training set contains image-cutouts (inputs) where the classes of each cutout is already known and is used to find rules, and the test set contains unseen cutouts used to evaluate the correctness of the learned rules.

There are a number of learning methods which can be applied to object classification [2][16]. Some of these include nearest neighbor, nearest centroid, neural network classifier, GP classifier.

A typical performance measure used in object classification is *classification accuracy* [14]. Classification accuracy is the number of correctly classified examples or object cutouts as a percentage of the total number of examples or object cutouts used in the classification process. The higher the classification accuracy, the better the classifiers performance.

2.4.3 Object detection

Object detection is a more difficult task than object classification as it involves not only object classification but also *localization* [16]. Localization involves finding (or detecting) objects of interest in an image. For multi-class object detection, there is an additional requirement of

then categorizing the detected object into a correct class from many different classes. In order to perform localization, object detection is done via a window-sweeping method which moves an input window across the image to find all objects of interest [16].

The Window-Sweeping Method for Feature Extraction and Object Detection

The window-sweeping method is a dimensionality reduction technique used to extract image features such as pixel statistics from the entire image. In most cases, the extracted information is written to a *pattern file* which represents a list of all the corresponding feature vectors. The window-sweeping method involves moving a fixed-size input window across an image, pixel by pixel, extracting image features at every location of the sweeping window [14]. The size of the input window is usually large enough to contain the largest object of interest in the image but also small enough as not so miss out on too much detail when extracting features. Every feature vector extracted at every location of the sweeping window is written to the patterns file, along with the location of the input window, usually in $< x, y >$ coordinate notation form. This method reduces the often large pixel map into a list of more useful feature vectors contained within the pattern file.

```
10 10
12.435 65.346 78.856 0.334
10 11
14.865 76.918 88.293 0.913
10 12
24.554 90.546 80.909 0.548
...
```

Figure 2.6: Excerpt from a pattern file with four features forming each feature vector

Object detection is also a more difficult problem than object classification in that it is often more computationally and time intensive. This is due again to the localization requirement. Where classification uses selected object-cutouts from one or many images (the number of cutouts usually corresponds to the number of objects in the image), detection regards every moving input-window in the sweeping procedure as an example or cutout. This results in the number of examples in the training set being vastly increased, even if only one training image is used.

A different performance measure is also required for detection as opposed to classification to meet the additional demand of localization [14]. A typical measure would include the detection and false alarm rate (see section 4.4.2), or recall and precision rate. In both examples, one measures how many true objects have been detected (detection and recall rates) and the other how many non-objects have been reported as true-objects (false alarm and precision rates). It is important to note that one measure is always traded-off for the other. That means, in trying to improve the detection rate or recall rates, there is often an increase in the false alarm or precision rates, or vice versa.

2.5 Genetic Programming for Object detection

Since genetic programming is relatively young learning/adaptive technique, it has only been applied to a limited number of object detection problems in recent years [16]. Presented below is a brief list of such object detection problems where a GP approach was

successful:

- *Automatic target recognition (ATR) systems*: Tackett and Char (1997) compared two GP approaches – the first using statistical features (extracted after some image processing) and the second using raw pixel values – against the detection results using a multilayer perceptron network and binary tree classifier, all applied to the same ATR problem using *US Army Night-Vision Terrain* images with target objects embedded in natural scenes [15].

- *Ship detection from satellite generated Synthetic Aperture Radar (SAR) images*: Howard, Roberts and Brankin (1999) used SAR imagery of the English Channel to detect ship objects in the ocean (background). They used pixels statistics such as the average and variance of group of pixels from a set of input regions chosen at various location around the images as features [8][10].

- *Finding blood haemorrhages and micro-anneurisms in retinal images*: Zhang (2000) compared a domain independent multi-class GP detection systems performance on a range of images including computer generated images (easy), scanned photographs of Australian 5c and 10c coins (medium difficulty) and retinal images containing blood haemorrhages and micro-anneurisms (very difficult) [16].

- *Detecting low-resolution 5c and 10c coin images*: Zhang, Andreae and Prichard (2002) explored a similar domain independent GP approach (as the one mentioned above), increasing the difficulty of the detection problems by decreasing the resolution of the image data [14].

Much of this work has focused the detection approach on *binary detection* problems and/or using a *multiple stage process* [9][10] [16].

The binary detection problem involves reducing the problem down into only two classes of interest: object and background. All images therefore only contain one class of object of interest and the background. This makes the task simpler as the genetic program solutions need not separate different classes of objects from each other, only object from background. Examples of such approaches are seen in [6][10]. This one-class approach is not sufficient given the increasing complexity of images and objects contained within these images in current object detection problems/tasks.

The multiple-stage process for object detection consists of four main phases: preprocessing, segmentation, feature extraction and classification. This approach focuses a large amount of attention on 'processing' or enhancing the initial data for the detection phase, which usually requires some domain knowledge specific to the data or task. As a consequence, the process becomes increasingly domain dependent, relying heavily on the the image processing techniques applied in first two phases. Another serious limitation to this approach is that each of the intermediate stages often depends on the results from the previous step. For example, if the segmentation phase was not particularly useful, the quality of features extracted will also poor resulting in the poor detection results.

A useful example to illustrate the usage of this approach is seen in *Genie*, a genetic programming based classifier used to detect complex terrain features such as golf courses and roads from remote sensing data sources [5][9]. They use extensive low-level image processing operators such as edge detectors, texture measures and morphological filters into enhance the original data (preprocessing and segmentation stage) before detection. This however relies heavily on geographical and geo-spatial knowledge built-in to the image processing algorithms.

2.5.1 Areas of Concern Using GP for Object Detection

Although GP has been successful for some detection tasks there are still many areas of concern using this approach. Some of these include:

- Very long training times, which can range from days to hours depending on the difficulty of the problem and the amount of data. This is due to every program in the population having to 'sweep' the entire training image (or images) for the objects of interest (repeated for every generation of programs).
- The evolved programs are often large, complicated and very difficult to interpret. They often contain many redundant sub-programs. These complicated programs also give the final impression that the evolution engine is a 'black box', as it becomes difficult to see exactly how the evolved programs solve a given problem.
- It is difficult finding the most powerful and useful set of image features. The GP engine searches through the set of given features finding the most useful subset to incorporate into the evolved programs. This contributes to the long training times as finding useful features in a large search space is time and computationally expensive.
- It is difficult finding a good fitness function. The fitness function is the only way of evaluating how useful an evolved program is. It must accurately rate good (fit) programs over poorer ones in its ability to solve the problem and effectively discriminate these poorer programs from being evolved over the better (fitter) ones. The fitness function must also be *continuous* ensuring any improvements in a programs fitness are noticed.
- The range of problems GP can successfully be applied to is an ongoing area of research. For example, can GP be used to solve problems where the objects of interest are of different sizes, there are many different classes of objects or the images contain highly-cluttered backgrounds?

2.5.2 Some Issues This Project Will Address

This project developes three different approaches to the detection tasks. The goal is to find ways to reduce the training times while still producing the best possible detection performance. The first approach involves training a population using a straight-forward sweeping-window approach, where entire image is scanned for objects of interest. The second method *applies* evolved program solutions trained using object classification to object detection in the sweeping-window style, as object classification involves a much shorter training process (its fitness cases are a selection of object cutouts, not the entire image). The goal here is to investigate whether these evolved programs are robust enough for the sweeping-window approach used in detection. The third approach involves further evolving an entire classification-trained population using the sweeping-window training approach (used in first method) with the goal of *refining* these classification-trained solutions for object detection.

This report will also investigate the effects of additional constraints to the fitness function. A multiple objective fitness function is developed aimed both at favouring the evolution of smaller programs over larger ones, and creating a more continuous search space for the GP evolutionary engine.

In addition, this project will address the effects of using different features sets. The goal here being to explore which features are more powerful for object detection than others.

All these research questions will applied to four different detection problems, investigating what range of problems the GP approach can effectively be used for.

Chapter 3

Tasks

To investigate the research questions described in section 1.1 and briefly in section 2.5.2, three image databases containing four object detection tasks of increasing difficulty are used in the experiments. All images have two or more classes of object of interest present. The sizes of the images from database 1 are 100x100 pixels whereas the sizes of the images from database 2 and 3 are approximately 600x600 pixels.

Task 1. Contains computer-generated images as shown in Figure 3.1. This is considered the easiest problem as these imaages consist of well defined objects of different shapes and intensities against a uniform background. There are two classes of objects interspersed around the image: light-grey circles (class 1) and dark grey squares (class 2). The size of the objects of interest are 18x18 pixels large.

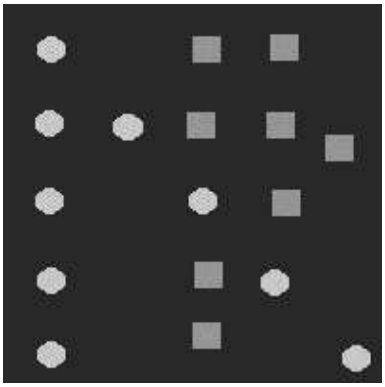


Figure 3.1: Computer-generated images (easy)

Task 2. Contains more real-world data and is intended to be more difficult, as shown in Figure 3.2 . It contains scanned images of New Zealand 5 cent and 10 cent coins against a plain and uniform background. There are two classes of different sized objects interspersed around the image: 10 cents (class 1) and 5 cents (class 2) coins. The size of objects of class 1 (10 cents) are 80x80 pixels and objects of class 2 (5 cents) are 60x60 pixels.

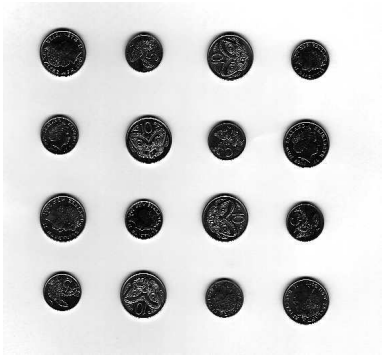


Figure 3.2: NZ 5c and 10c coin images (medium difficulty)

Task 3. Database 3 is intended to be more difficult than the previous, as shown in Figure 3.2. It also contains real-world scanned pictures of New Zealand 5 cent coins against a noisy background. There are also two classes of interest: tails (class1) and heads (class1). The added difficultly lies in that the different classes of objects are harder to distinguish being of the same size and shape, and the background is very noisy (more like real-world generated images). Both objects of interest are of size 60x60 pixels.

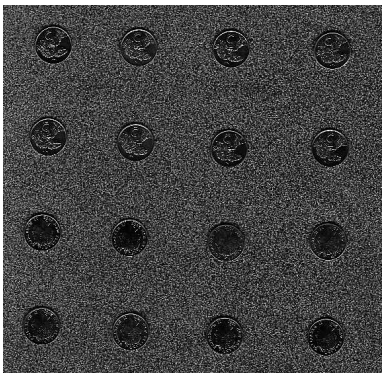


Figure 3.3: NZ 5c heads and tails on noisy background (very difficult)

Task 4. The forth task involves an extension of task 2, and is intended to be the most difficult of all (same images is figure 3.2). Here the objects have been split up into four classes: 5c tails (class1), 5c heads (class 2), 10c tails (class 3) and 10c heads (class 4). The difficulty lies not only in separating 10c from 5c, but also heads from tails in both cases.

3.1 Training and Test Set Size

For each of the detection tasks mentioned above, the images in the databases were split into a training and test set. Due to the long training times for detection process, the training set was chosen to be smaller than the test set.

For the SFD and ROCD methods: the training set contained one image for each task, and the test set contained 3 unseen images for each task.
For the OCAD method: the training set contained three images for each task, and the test set contained 3 unseen images for each task.

Chapter 4

Methodology

This chapter in outlines the methodology, implementation and experiment settings according the research goals outlined in section 1.1. This chapter is split into five main sections where each is organized as follows:

Section 4.1 Describes the three different approaches developed for the detection tasks,

Section 4.2 Describes the GP function and terminal sets being investigated in this project,

Section 4.3 Describes the classification strategy used in the different approaches,

Section 4.4 Introduces the two different fitness functions being investigated, and

Section 4.5 Outlines the evolutionary parameters and termination criteria.

4.1 Three approaches to Object Detection

Three methods were considered and developed for this object detection in this project: straight-forward detection, object classification applied to detection and refining object classification for detection. Each is discussed below.

4.1.1 Method 1 - Straight Forward Detection (SFD)

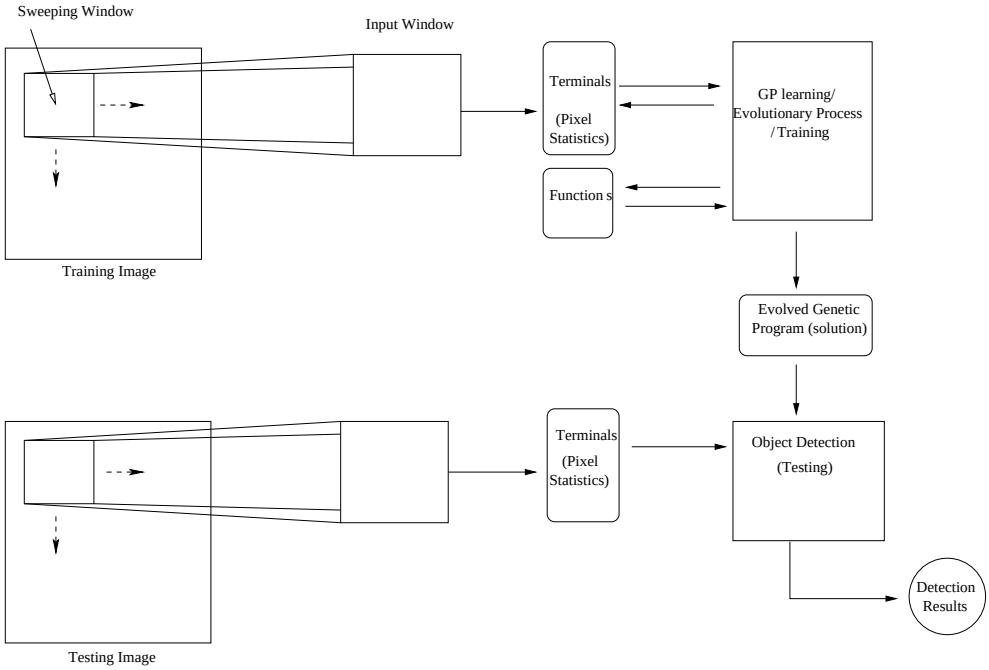


Figure 4.1: Overview of straight-detection approach

Figure 4.1 shows an overview of the *Straight Forward Detection* (SFD) process. It consists of a learning and a testing phase. In the learning/evolutionary phase, an input window large enough to fit a single object is moved across the training image, in a sweeping-window fashion, to detect all the objects of interest. Image features are extracted from each of these moving windows and used as input by the GP evolutionary engine. The evolutionary process consists of generating many different programs (possible solutions), applying these programs to the image to obtain their fitness (how good they actually are) and then evolving these programs using the GP operators. This process is repeated over a number of generations until the optimal solution is found or other termination criteria are met. That is, a program that successfully locates all the objects of interest in an image, without reporting any false alarms (non-objects).

The testing phase applies the best evolved genetic program obtained from the learning process to a test set which consists of a number of unseen images, to measure the detection performance.

4.1.2 Method 2 - Object Classification *Applied To* Detection (OCAD)

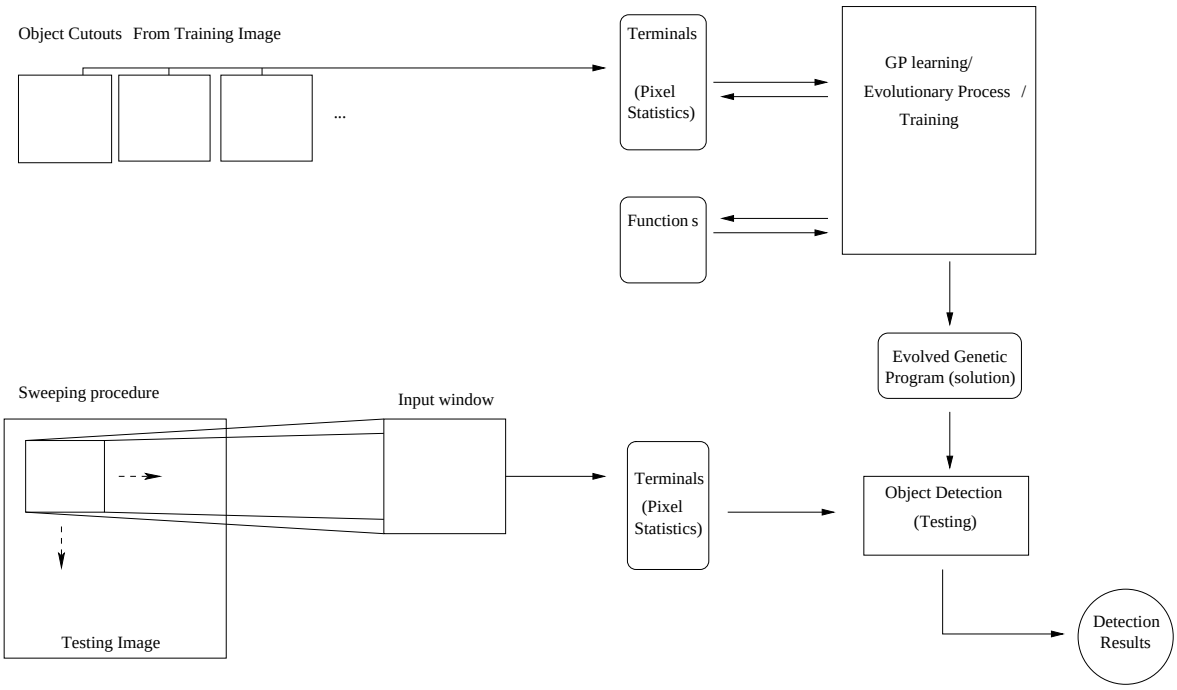


Figure 4.2: Overview of classification applied detection approach

Figure 4.2 shows an overview of using the object classification *applied to* detection (OCAD) process. In a similar fashion to *straight-forward* detection, this also consists of a learning and testing phase. The main difference however, is in the learning phase. Here the GP system is evolved using *object cutouts* of each class, rather that the entire image. When programs are initially generated or evolved using the GP operators, they are applied to all the object cutouts to evaluate their accuracy (fitness), not the entire image as in the SFD method . With this change in learning cases (fitness cases) also comes a requirement for a different fitness evaluation compared to the SFD method. Here programs are evaluated on how many of the object cutouts they are able to correctly classify or their classification accuracy (section 4.4.2), opposed to detection and false alarm rates .

The testing phase is the same as straight-forward detection. The best evolved genetic program obtained from the learning phase is applied to the same test set as used in the SFD method to measure the detection performance.

The motivation behind this method is that it greatly reduces training times as there are a significantly smaller number of fitness cases. The fitness cases correspond directly to the number of object cutouts, whereas the number of fitness cases for straight-forward detection is typically very large (roughly equal to the cross product of the image width and height minus the input-window size). With this vast reduction in training time also comes the opportunity for the GP system to include multiple images in the training set, something that straight-forward detection makes very difficult due to the computational cost. This is advantageous as it implies the evolved program solutions will be more generalized having seen more images while training (less chance of over-fitting).

A limitation would be in selecting the most useful and appropriate cutouts for GP classification. This is important as it will determine the classifiers ability to generalize, only having trained on a limited number of cutouts. In this investigation, object cutouts from

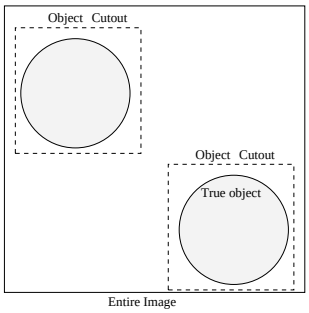
three images were used in training. These include all objects of interest for all classes, and a number of randomly and specially selected background objects.

Method of Selecting and Extracting Cutouts

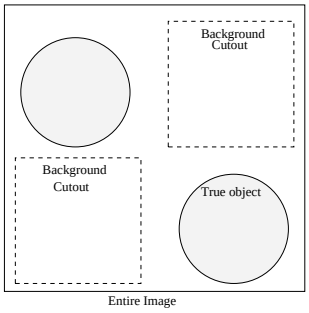
The selection of object-cutouts to be included in the training set is important as it not only affects how well the programs learn to separate the different classes, but also how well they will perform when applied straight to the detection process (on entire unseen images). The cutouts must therefore be inclusive of all the centers of the objects of interest, as well as sufficient background cutouts to cover all cases where the input window is not centered over a true object. These include cases where there is a combination of background and a portion of some object in the input window. For example, when there is half an object in the input window.

The method of four categories of cutouts were selected and extracted from the as training example for object classification:

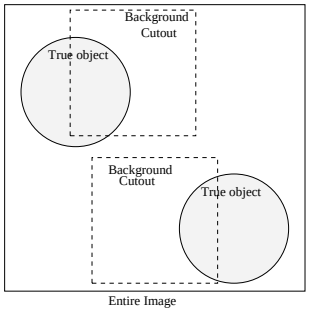
1. Extract *object* cutouts where the input window is centered over all the objects of interest, depending on the class of the object. For example, the image below will have two object cutouts.



2. Extract *background* cutouts where the input window is centered over background only. The image below will have two background cutouts.



3. Extract *background* cutouts from specially selected centers where the input-window includes a combination of background and true objects, usually involving some portions of the true objects. The image below will have two 'more difficult' background cutouts.



4. Depending on the number of fitness cases require, randomly select n more *background* centers from anywhere else in the image that does not correspond to any centers chosen from step 1, 2 and 3.

4.1.3 Method 3 - Refining Object Classification for Detection (ROCD)

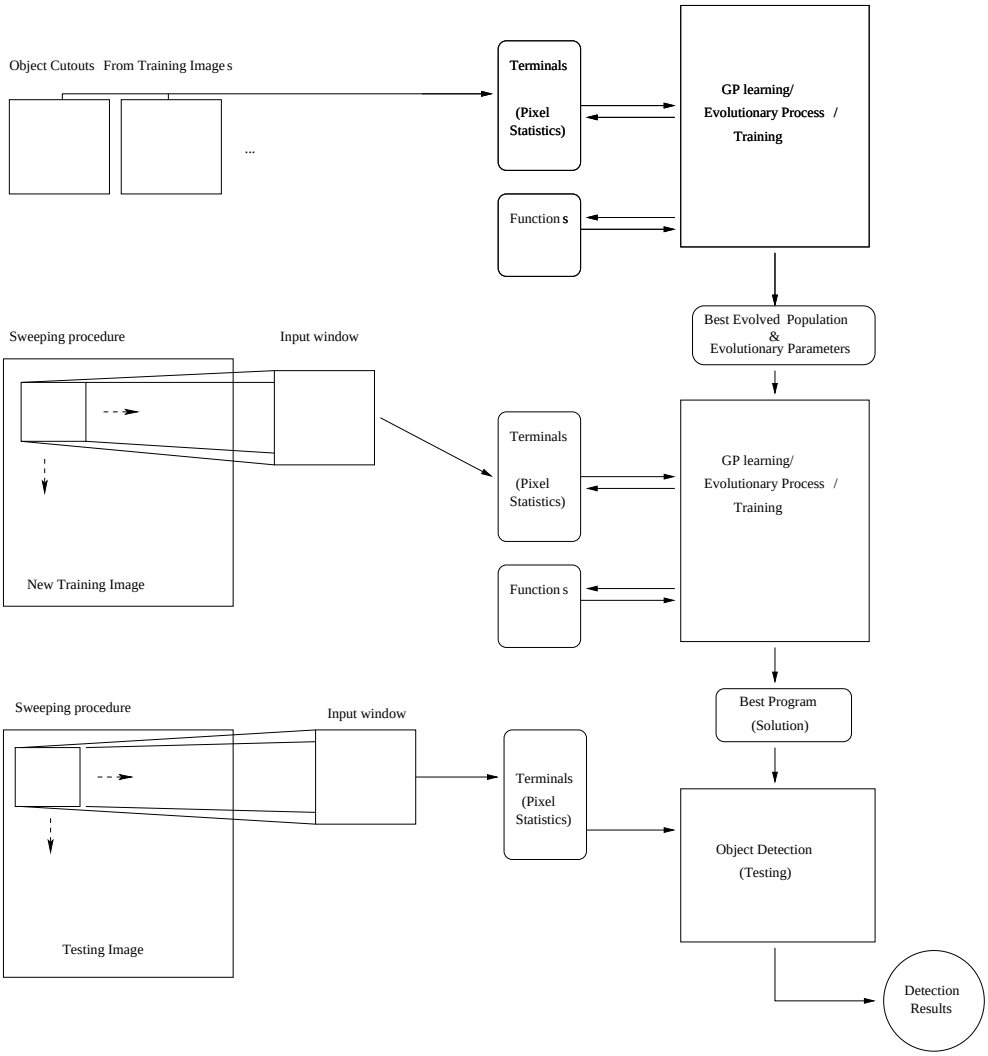


Figure 4.3: Overview of *refining* approach

Figure 4.5 shows an overview of the refining object classification for detection process (ROCD). The difference, compared to the previous two approaches, lies in the *two-phase* learning process. Instead of applying the evolved solutions from the OCAD method directly on the entire test images to obtain the detection results, the entire evolved population is fed into a secondary learning phase for further evolution. This secondary learning phase then uses the SFD method to improve the detection performance by further evolving the population.

The testing phase then applies the best evolved genetic program obtained from the second-learning process to the test set to measure the object detection performance.

This hybrid method has two advantages:

1. Although the secondary learning phase uses a large number of fitness cases (a limitation of the SFD method), the best solution should converge faster as the initial population is not random (like straight-forward object detection) but an already evolved population.

2. If the best evolved program from the OCAD method does not perform well when applied to an entire unseen image (testing), it has the opportunity to improve its fitness via further training. This is attractive as teh OCAD method could be a useful alternative to the SFD method in saving on training times.

A limitation however is that if the task is not easily solvable by classification and the average fitness of the evolved population is low, then the secondary learning phase will require a large number of generations to evolve these poorer programs for detection.

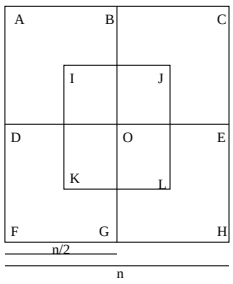
4.2 Functions and Terminals

4.2.1 Terminal Sets

Terminals form the inputs to the GP system. In this project they correspond to a number of image features and a user-defined threshold value T. These features are calculated from pixel statistics based on the direct pixel values. Specifically, the mean and standard deviation of a group of pixels in a region. This represents the overall brightness and contrast of the region in the image. Pixel statistics make good features for rotational and translational invariance. We do not investigate size invariant object detection in this project.

The method of extracting these features involves breaking the moving input-window into smaller regions, as local region features have proved more effective than whole region features [17]. This project involves evaluating the effects of five terminal sets each using different local regions. These were split into square-region features (Term Sets I, II, III) and circular-region features (Term Sets IV, V). These terminal sets are described in detail below:

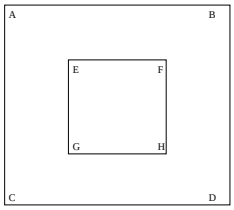
1. **Terminal Set I** - This set computes the mean and standard deviation of the whole sliding window, the outside four quadrants of the input-window and the central quadrant, as shown below. In total, the rectilinear regions pixel statistics make up 12 features.



Mean	SD	Regions
F1	F7	Big square A-C-H-F
F2	F8	Upper left square A-B-O-D
F3	F9	Upper right square B-C-E-O
F4	F10	Lower left square D-O-G-F
F5	F11	Lower right square O-E-G-H
F6	F12	Small central square I-J-K-L

Figure 4.4: Terminal set I - 5 rectilinnear quadrants

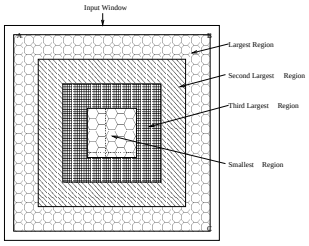
2. **Terminal Set II** - This set computes the mean and standard deviation of the whole input-window and the central square. The motivation for this terminal set was to test the effectiveness of the four quadrants. Without it, the search space for the solution becomes smaller, thus simplifying the problem. These use a total of 4 features.



Mean	SD	Regions
F1	F2	Big square A-B-C-D
F3	F4	Small central square E-F-G-H

Figure 4.5: Terminal set II - 1 rectilinnear quadrant

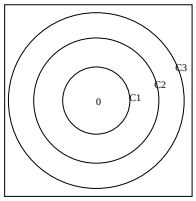
3. **Terminal Set III** - This set computes the mean and standard deviation on *whole* square-regions, each increasing in size (4 regions), centered in the middle of the input window.



Mean	SD	Regions
F1	F2	Largest region
F3	F4	Second Largest region
F5	F6	Second Smallest region
F7	F8	Smallest region

Figure 4.6: Terminal set III - 8 square-region features

4. **Terminal Set IV** - This set computes the mean and standard deviation on a series of three concentric circular regions centered in the input window. Each increases in size, where the larger circles overlap the smaller circles.

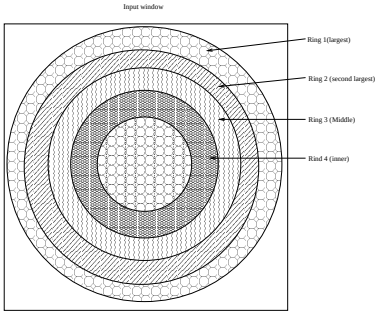


Mean	SD	Regions
F1	F4	Big circle C3
F2	F5	Medium size circle C2
F3	F6	Small size circle C1

Figure 4.7: Terminal Set IV - 6 circular features

5. **Terminal set V** - This set computes the mean and standard deviation on the *rings* of

four concentric circular regions centered in the input window. Each ring does not overlap with any another ring.



Mean	SD	Regions
F1	F2	Ring 1 region (Largest ring)
F3	F4	Ring 2 region (second largest ring)
F5	F6	Ring 3 region (middle ring)
F7	F8	Ring 4 region (inner ring)

Figure 4.8: Terminal set V - 8 circular features

4.2.2 Function Set

The function set forms the non-terminal nodes of the constructed genetic programs. It consists of the four standard arithmetic operators and a conditional operator.

$$FunctionSet = \{+, -, \%, *, if\}$$

The +,- and * operators have their usual meanings (addition, subtraction and multiplication) while % means *protected* division. That is, usual division except that a divide by zero gives a result of zero. Each of these operators take two arguments and return one. The `if` function takes three arguments. If the first is positive, the second argument is returned; otherwise it returns the third argument. This allows a program to contain different expressions in different regions of the feature space.

4.3 Object Classification and Detection Strategy

The output of the genetic programs in this approach is a floating point number. In conventional single-class or binary object detection tasks, the division between positive and negative numbers of a genetic programs output corresponds to the separation of the objects of interest (single class) and the background [17]. Since this project involves multiple class object detection, where two or more classes of objects of interest are concerned, this strategy must be extended to accommodate for many classes versus background.

In this approach, both object classification and detection is done using a *program classification map*, as shown in figure 4.9 [19]. From the output of a genetic program, all values that are less than or equal to zero are classified as background while those greater than zero are an object. The separation of the different classes is determined according to the different thresholds in classification map.

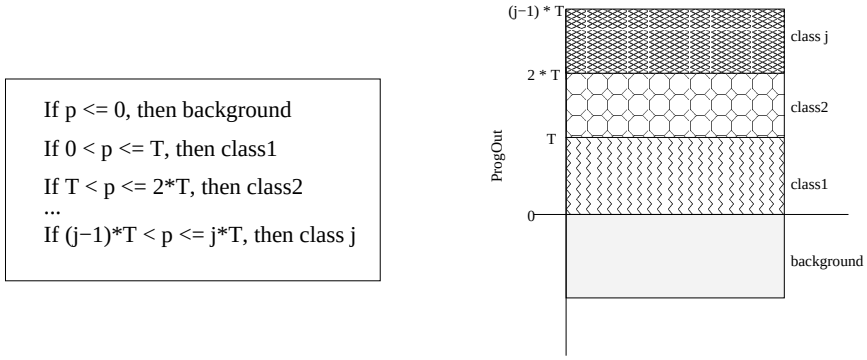


Figure 4.9: Classification map

From the map, p refers to the output of genetic program, j to the number of classes of interest and T to a user-defined constant which plays the part of a threshold. For example, if a program output p at a particular input window location is 21.2345 and T is set to 10, the object in the input window will be classified as class 2:

$$\frac{p}{T} = \frac{21.2345}{10} = 2$$

Therefore, using `T` and a programs output p at all locations in the window-sweeping procedure will determine whether an input-window location is considered an object, and if it is, what class it belongs to from class 1, class 2, ... , $class_j$.

Using a classification map such as this one therefore frames the object detection and classification problem as a special case of solving a non-linnear regression problem. The function being learned is the function that separates all objects of interest from each other and from the background. The variables correspond to the GP terminals and the arithmetic operators to the GP functions.

4.4 Fitness Measure

The fitness function is one of the most important components in genetic programming as it allows the GP system to evaluate how well a program performs on a task. In the context of this project, the fitness function directly measures how well programs perform on the detection or classification tasks. Since both detection and classification methods are being investigated, two fitness measures are thus required: one for detection (approaches 1 and 3) and the other for classification (approaches 2 and 3). The fitness function for object detection is designed to evaluate programs according to the detection and false alarms rates together, both important to the overall performance as detection requires both object classification *and* localization. Straight object classification on the other hand, evaluates programs according to their classification accuracy, where detection and false alarm rates no longer apply.

4.4.1 Fitness Functions for Detection

The fitness for object detection, as mentioned above, is calculated based on the detection rate and the false alarm rate. The detection rate (DR) is the number of true objects correctly reported by the detection system as a percentage of the total number of known or actual objects-of-interest in the image. The false alarm rate (FAR) is the number of non-objects (background) incorrectly reported as true objects as a percentage of the total number of known objects-of-interest present in the image [19].

For multiple-class object detection training on a single image, the DR and FAR are:

$$DR = \frac{\sum_{i=1}^{n} N_{true}(i)}{\sum_{i=1}^{n} N_{known}} \times 100\% \tag{4.1}$$

$$FAR = \frac{\sum_{i=1}^{n} N_{reported}(i) - \sum_{i=1}^{n} N_{true}(i)}{\sum_{i=1}^{n} N_{known}} \times 100\% \tag{4.2}$$

Where:

- m is the number of classes of interest,
- N_{true} is the number of true objects correctly reported for class i,
- N_{known} is the number of actual (or known) objects for class i,
- $N_{reported}$ is the number of objects reported for class i.

The goal is to achieve a high detection detection rate and a low false alarm rate. That means, locating and reporting 100% of the known objects with 0 false alarms (0% FAR). There is however a tradeoff between these measures. Often trying to increase the detection rate often results in an increase in the false alarm rate, and similarly, trying to lower the false alarm rate can result in a decrease in the detection rate [19] .

Note a few points on the FAR. Since there are usually many more background 'objects' present in an image than true-objects, the false alarm rate can be greater than 100% under the above definition. In such cases where the FAR is greater than 100%, it implies that the detection system incorrectly reported more non-objects as true objects than there actually are. False alarm rates also tend to escalate to very large values even in simple detection problems just because of the sheer number of non-objects to objects ratio in an image. It is not uncommon to have rates ranging between 200% and 2000% for extremely difficult detection problems such as those involving objects-of-interest surrounded by very cluttered backgrounds.

For example, consider an image of size $100x100$ pixels – which is small considering a typical 'real world' image can easily be of size $1000x1000$ pixels – containing 20 objects of interest of a single class, all of size 10x10 pixels. The number of known centers N_{known} will correspond to the center co-ordinates of each these objects of interest in the image, totalling 20, whereas the non-objects will be all the other co-ordinates which are not objects, totalling 8080, from:

$$(100 - 10) \times (100 - 10) - 20 = 8080$$

This illustrates the high *object to non-object* ratio of 20:8080 for even a relatively small image.

It is important to mention that all such non-objects could represent a possible false alarm in a detection system. This makes the task of object detection difficult. Consider also the case where the true center of an object in an image is marked at co-ordinates (x,y). According to the false alarm rate, the immediately neighboring pixel at co-ordinate (x+1,y) is a false alarm. Intuitively this does not seem correct as both these co-ordinates belong to the same object in question yet a detection system will mark (x+1,y) as a false alarm when clearly it is not.

For this reason, the detection and false alarm rates are calculated using a **clustering approach** [17]. This effectively clusters neighboring co-ordinates together as a single object. The size of the clusters are defined according to a TOLERANCE value, where TOLERANCE is a user-defined constant. The clustering approach therefore implies that if a detected center occurs within TOLERANCE pixels of a cluster, it belongs to that cluster. The detection and false alarm rates using the clustering approach are as follows:

1. *Detection rate* : Neighboring pixels around the true-center of an object will not be counted as false alarms as they fall within TOLERANCE pixels of a known center. A consequence is that the detection system will no longer report the 'true center' of an object, but rather the first *detected center* it finds corresponding to that object cluster.

2. *False alarm rate* : If a false alarm is reported (non-object), then all other possible false alarms reported within TOLERANCE pixels of that original false alarm is instead clustered as only *one* false alarm. An object is therefore only reported as a 'detected object' if it is not within TOLERANCE pixels of another detected object (cluster).

Using clustering effectively translates the high false-alarm-rate problem into a more viable measure as the false alarm pixels are instead treated as false alarm 'objects'.

Recall that one of the goals in this project are to measure the effect of different fitness functions on program evolution. For this reason two fitness functions are used in the detection tasks and will be evaluated against each other to investigate their effects.

1. The first fitness function is the typically used fitness function for object detection based on the detection rate (DR) and false alarm rate (FAR) [19]. K1 and K2 are user-defined constants that reflect the relative importance between the detection rate and false alarm rate. These are treated as addition parameters to the GP system (section 4.5). This fitness function will be zero for the idea case.

$$fitness(DR, FAR) = K1 * (1 - DR) + K2 * FAR \tag{4.3}$$

2. The second fitness function was developed for the purposes of this project. It uses the DR and FAR in the same way as the fitness function above, with two additional constraints: false alarm area (FAA) and the Program Size (Prog_size). This is considered a *multiple objective* fitness function as it rewards programs that favour all of the constraints. Similar to the use of constant weight values seen above (equation 4.3) K1, K2, K3 and K4 reflect the order and relative importance of each of the four constraints.

$$fitness(DR, FAR, FAA, ProgSize) = K1*(1-DR)+K2*FAR+K3*FAA+K4*ProgSize \tag{4.4}$$

The rest of this section describes the rationale behind adding *FAA* and *ProgSize* to the fitness function, outlines the method of applying the fitness function to object detection and introduces the fitness function used for object classification.

Program Size in the Fitness Function

Program size is included in the fitness function for the end goal of evolving programs that obey the *law of parsimony*. Parsimony is a term borrowed from physics and philosophy which means that the simplest between two or more competing theories is usually the one that is preferred. In terms of program evolution, the simplest program detectors which achieve a good detection performance will be favoured for evolution. For example, if two programs have the same detection performance, the smaller program will be fitter.

Program size is measured by the number of functions and terminals that make up a single genetic program. For example, program (a) below has *ProgSize* of 7 and (b) has a *ProgSize* of 13.

```
(a) (*(+ x x)(- y y))
```

```
(b) (-(*(+ x x)(- y y)) (+ x (* y x)))
```

The desired effect of building parsimony into program evolution is so that populations do not end up containing individuals overly complex in structure. This is a common problem with genetic programming as many 'redundant' subtrees are often built into evolved programs via the crossover or mutation operators. Redundant here means subtrees that either do not contribute to the overall fitness of the program or subtrees that can be simplified down further. For example, subtree `(/(* x x) x)` can be simplified to just `x`.

Simpler programs are desired for two important reasons: efficiency and understandability. Smaller programs take a shorter time to evaluate than longer programs, and if dealing with large populations with 1000 individuals or more, this can lead to a considerable amount of time saved in evaluating these programs. The second reason, understandability, is for our benefit as scholars of such learning paradigms. It aims to remove the 'black-box' feel associated with genetic program evolution. This is useful as, in order to observe and understand the validity of genetic programming as a machine learning technique, we need to be able to see exactly how it has 'learned' the solution to a task and how we may improve on it. Parsimony is also known as the theory of Ockham's razor [15]

As a consequence of using program size in the fitness function, the best (optimal) fitness is no longer 0. This is because any programs size will always be an integer greater than 0. Instead the *evolutionary termination criteria* was modified to accommodate for this by allowing the best program to have an optimal fitness only if the following condition is specified

$$fitness - (K4 * ProgSize) \leq 0$$

.

False Alarm Area in the Fitness Function

The false alarm area (FAA) is an object detection constraint introduced by [17]. It corresponds to the number of unclustered centers generated by a program when calculating the false alarm rate (FAR). Recall that while the clustering approach used to calculate the FAR does not count those false alarms that fall within TOLERANCE pixels of a clustered center, FAA does. For that reason, FAA is calculated using the same definition (*equation* 4.2) given for false alarm rate FAR (in section 4.4.2), except the reported centers are not clustered.

The motivation for including FAA in the fitness function is that it can measure small improvements in performance, something which just FAR can not. This is important as otherwise the evolutionary process will have difficulty in selecting better programs. Figure 4.10 demonstrates how FAA can be beneficial in measuring small changes to a program.

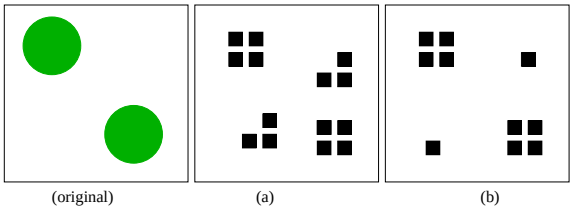


Figure 4.10: Detected centers with different false alarm area rates

From figure 4.10, (a) and (b) are object detection maps produced by two different programs using the *original* image. Although both a and b detect both true objects (100%) they also report two false alarms. If TOLERANCE pixels allowed is 4, then according to *equation*

4.3 they are equally fit with 100% DR and 100% FAR (2 clustered centers as a percentage of known centers). But this does not reflect the fact that clearly program b is better than program a because it produced less false alarms. Using FAA would reflect this slight difference in performance and rank these programs accordingly as,

$$a_{FAA} = (6/2) * 100 = 300\%$$

$$b_{FAA} = (2/2) * 100 = 100\%$$

Applying the Fitness Function to Object Detection

The fitness of a genetic program for detection is calculated as follows:

1. Apply the program as a moving window template to the training image to obtain a program output value for each location (x and y co-ordinates). According to the classification map (from figure 4.9), label each location with the object detected.

2. Find the centers of these detected objects by the clustering algorithm (within a TOLERANCE number of pixels).

3. Match these detected centers with the known locations of each of the true objects to find the detection rate. Using these results, calculate the false alarm rate and if relevant, the false alarm area and program size.

4. Calculate the fitness of the genetic program according to *equation 4.4*.

4.4.2 Fitness functions for classification

The fitness function for object classification is based on *classification accuracy*, which in defined as follows:

$$accuracy = \frac{\sum_{i=1}^{n} N_{correct}(i)}{\sum_{i=1}^{n} N_{total}} \tag{4.5}$$

Where,

1. n is the number of classes of object cutouts,

2. $N_{correct}$ is the number of correctly classified object cutouts according to class i,

3. N_{total} is the number of cutout objects for class i.

The fitness function using classification accuracy (CA) in defined as:

$$fitness(CA) = 1 - (CA) \tag{4.6}$$

Like the fitness function for detection, the best program in object classification will also have a fitness of 0. The method of evaluating a programs fitness is also similar, except that the given program is applied to all the cutouts, not the entire image to determine the classification accuracy. Only this fitness function is considered for classification as the goals for this project focus on trying to methods to improve object detection, not classification. This fitness function is simple and effective and will suffice for the role classification plays in this research.

4.5 Evolutionary Parameters and Termination Criteria

The parameters used in this approach are presented in table 4.1. These were split into three main parameter types: *search, genetic* and *fitness* parameters.

Types	Parameter names	easy Images	Coin Images
Search Parameters	Minimum depth for programs	2	2
	Maximum depth for programs	6	8
	Maximum generations	100	200
	Individuals in population	800	2000
	input window size	20x20	72x72
	Fitness function	I & II	II
Genetic Parameters	Mutation Rate	29%	29%
	Crossover Rate	70%	70%
	Elitism Rate	1%	1%
Fitness Parameters	T	100	80
	K1	5000	5000
	K2	100	100
	K3	10	10
	K4	1	1
	Tolerance	8	30

Table 4.1: Evolutionary parameters for detection tasks

The evolutionary process was terminated either when the number of generation reached the pre-defined number *max-generations* or when a programs fitness was ideal (0 in most cases).

Chapter 5

Results

This chapter presents all the results from the experiments and research questions outlined in the goals section (section 1.1). The results are summarized and analyzed and include: the detection performances from the three different approaches, the feature set performance on the different tasks, and the fitness function evaluation on each of the four tasks. This chapter is split into five main sections and each is organized as follows:

5.1 Results from the computer-generated images (task 1)

5.2 Results from the 10c vs 5c coins (task 2)

5.3 Results from the 5c heads vs 5c tails coins (task 3)

5.4 Results from the 5c heads vs 5c tails vs10c heads vs 10c tails (task 4).

5.5 Summary of the fitness function comparison, the detection performances using each method on the four tasks, and terminal set performance evaluation for each the four tasks.

5.1 Task 1 — Generated Easy Images

This section presents the detection results using the straight-forward detection method. Since these results are ideal and achieved in relatively short training times (due to the small image size), **only** this detection method was used. Three runs were performed using fitness function I (*equation 4.3*) for each of the five terminal sets and the solutions applied to the test set to evaluate the averaged detection results (presented in table 5.1).

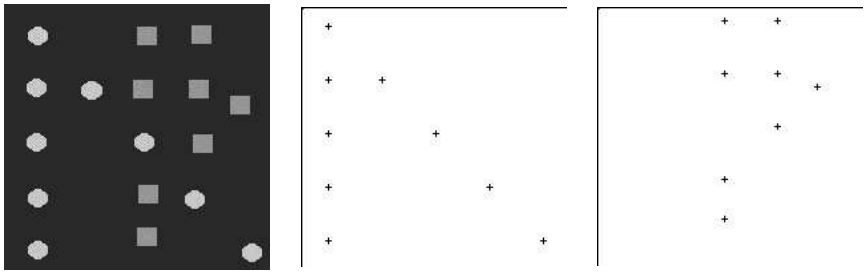
Terminal Sets	DR			FAR		
	class1	class2	**Total**	class1	class2	**Total**
TermSet I	100	100	100	0	0	0
TermSet II	100	100	100	0	0	0
TermSet III	100	100	100	0	0	0
TermSet IV	100	100	100	0	0	0
TermSet V	100	100	100	0	0	0

Table 5.1: Detection results for easy images

Table 5.1 shows that the detection system produced ideal results for the easy generated images. All five terminal sets achieved a 100 % detection rate for both classes of objects

with no false alarms. On average terminal set II found the ideal solution first, in about 15 generations. Terminal set III found the ideal solution in approximately 25 generations while Terminal set I took the longest training time, usually about 50 generations. This suggests that a simple simplification of the search space (less features) makes the solution easier to find. In all cases, the training times were approximately 30 to 90 minutes.

Figure 5.1 presents a sample object detection map to illustrate the detection results. It corresponds to a run using terminal set IV. Figure (a) is the original unseen image, and (b) and (c) are the detection maps (a black cross represents a detected object center for circles and squares).



(a) (b) (c)

Figure 5.1: Detection map for the easy images

5.1.1 Programs and Fitness Functions

This section investigates the effect of using the multi-objective fitness function in program evolution, specifically, the effect of using program size in the fitness function.

Recall that the two **Fitness functions** being compared are:

I. Fitness(DR, FAR) = A * (1-DR) + B * FAR

II. Fitness(DR, FAR, FAA, Program_Size) = A* (1-DR) + B * FAR + C * FAA + D * Program_Size

Using program size in the fitness function (II) yielded smaller program solutions on average than program solutions generated without using the program size in the fitness function (I). This was discovered by comparing the sizes of the evolved programs trained using both fitness functions on the *easy images* using terminal set III (square region features). The *easy problem* was the best of the tasks to evaluate this comparison of fitness functions because of the considerably shortened training time in comparison to the *coins problems*.

From a total of 80 independently evolved program solutions, all achieving perfect results on the training and test tests, half (40) were trained using fitness function I and the other half (40) using fitness function II. The average sizes were as follows:

- Fitness function I: generated average program size of 62.
- Fitness function II: generated average program size of 24.

Programs evolve using fitness function II appears practically beneficial as these smaller solutions allowed the evolution to converged faster, reducing the overall training time (in terms of generations taken to evolve). These programs are easier to interpret and contained less redundant building blocks (subtrees) than the programs evolved using fitness function I which takes much time and resources to compute.

The smallest evolved program using fitness function II was of size 16. One of these programs is presented below for a comparison.

```
(d/ (if (d/ (d- F7 T) F7) F8 (d* (d- F7 F3) F2)) (d/ F7 F7))
```

This is very much easier to interpret than a program generated using fitness function I, such as the one presented below which is of size 73 (slightly larger than the average size)

```
(d/ (d+ (d* (d/ (d/ F8 T) (d+ T F1)) (d+ (d+ T F2) (if F7 T F8))) (if
(d- (d* T F4) F5) (d- (d+ F6 T) (d* T F7)) (d/ (d- T F8) (d- F1 T))))
(d+ (if (d/ (d/ F2 T) (d* T F3)) (d+ (if T F4 F7) (d- T F5)) (d* (d/
F1 F6) (d+ F7 T))) (d* (d/ T (d+ T T)) (d/ T (d/ T F4)))))
```

A manually simplified version of the shorter program from above, after removing three redundant sub-trees, such as replacing the last term `(d/ F7 F7)` with 1 and replacing `(d/ (d- F7 T) F7)` with `(d- F7 T)`, is shown below. It is now of size 10.

```
(if (d- F7 T)
    F8
    (d* (d- F7 F3) F2))
```

Intuitively this solution is easy to interpret. First recall that the terminal values are calculated from the mean and standard deviation of the raw pixel-value intensities (I) (see Figure 4.3). Since class 1 (circles) is lighter in intensity than class 2 (squares), it follows that $mean\ (class1) < mean\ (class2)$, where $mean\ (class1)$ is approximately 120 and $mean\ (class2)$ is approximately 200. Yet both $mean\ (class1)$ and $mean\ (class2)$ are lighter than the background intensity, which is very close to 0, so it follows that $mean\ (class1)$ and $mean\ (class2) > mean\ (background)$.

According to the learned function, if part 1 (`if` condition) evaluates positive , the second term `F8` is chosen, else the last term is chosen. Since $mean\ (background)$ is less than threshold `T` (100), this condition will effectively only evaluate true when the moving input-window is over an object of interest, as F7 corresponds to the smallest square-region mean and both $mean\ (class1)$ and $mean\ (class2)$ is greater than `T` . In the case where the input window is over the background, the last term will be returned as the program output. Recall that an object is only considered found if the program output is greater than 0 (from classification map). The program ensures this always happens by constructing the the last term such that it will always return a value less than or equal to 0 if the input window is over background from following two cases:

1. If the input window is over background only, both means F3 and F7 will be the same, therefore evaluating to zero.

2. If a portion of object is visible in the input window with the rest as background, F3 will be greater than F7 as the mean of the larger window (F3) will increase due to the larger I values contributed from the portions of the lighter object. Thus returning a negative number.

This shows that this simple `if` function is enough to capture the difference between background and objects-of-interest based on their I values. If it is a object of interest, the standard deviation (SD) `F8` distinguishes between either class 1 or 2 where $SD\ (class1)$ is less than `T` and $SD\ (class2)$ is greater than `T`. Such analysis on the solution removes the

'black-box' feeling commonly associated with the programs generated in the evolutionary process, as we encountered in such programs as the second presented above (evolved using fitness function I). It allows the chance to dissect the construction of a program solution to see exactly what each component is responsible for, giving a more intuitive feel into how the program detectors works.

Therefore building the idea of parsimony – the simplicity of solution structures – into the fitness function is beneficial in finding solutions fast.

5.2 Task 2 — 10 Cent and 5 Cent NZ Coin images

The results for this task use all three detection methods, *straight-forward detection*, *object classification applied to detection* and *refining object classification for detection* (introduced in section 4.1), to evaluate their effectiveness. The subsequent sections present the detection results (detection and false alarm rates) achieved for each of the three methods using each of the five terminal sets, the best evolved programs and sample detection maps. The tables corresponds to how well each terminal set performed on the detection task, where the 'total' columns are the overall detection rates and false alarm rates.

These programs were generated using the evolutionary parameters from the table presented in section 5.5 and fitness function II (*equation 4.4* in section 4.4.1). Fitness function II was chosen over fitness function I because of the positive effects it was shown to have on program solution structures [section 6.1.1].

5.2.1 Results using Straight Forward Detection (SFD)

Each training run was executed three times for every terminal set, producing 3 solutions each to be evaluated against the test set. The test set consisted of 3 unseen images. The *averaged* detection and false alarm rates presented below are calculated from all three evolved solutions performance on the test set.

Terminal Sets	*DR*			*FAR*		
	class1	class2	**Total**	class1	class2	**Total**
TermSet I	100	100	**100**	0	0	**0**
TermSet II	100	100	**100**	0	20.83	**10.42**
TermSet III	100	100	**100**	0	12.5	**6.25**
TermSet IV	100	100	**100**	0	0	**0**
TermSet V	100	100	**100**	0	0	**0**

Table 5.2: Straight-forward detection results for 10c and 5c coins images

Only three out of the five terminal sets achieved perfects results: terminal sets I, IV and V. Although terminal set III did report some false alarms on the test set, this number is relatively low (10.42%). The reason being, that out of the three evolved program solutions, only two were ideal while the third was not, reporting some false alarms. This resulted the overall FAR average being 'bumped up' just slightly. In the case of terminal set II only one out of the three evolved solutions was ideal, the other two both reporting some false alarms.

Training Times

Another major difference between these performances was the times taken to find a solution. The averaged number of generations required to find a solution for each terminal set is given below for a comparison. This comparison is important as since at least one evolved program from all the terminal sets were able to achieve perfect results, the next best measure on the effectiveness of the different terminal sets is how *well* each terminal set affected the *convergence* of a solution.

Terminal Sets	Average generation for solution to converge	average time (hours)
TermSet I	34	26.5
TermSet II	128	51.5
TermSet III	96	38.5
TermSet IV	19	17
TermSet V	11	12.5

Table 5.3: Training times for 10c and 5c coin images using straight-forward detection

These results are a good indication of which terminal sets supported the fast convergence of program solutions. Terminal set V (circular-ring region features) found a solution the fastest – each of the the three programs was evolved in under 12 generations. The next fastest average training time was from terminal set IV (whole circular region features), taking around 19 generations to converge. The three square-based feature region terminal sets (I, II and III) all produced relatively high training times, with terminal set II the worst at an average of 128 generations for a solution to converge.

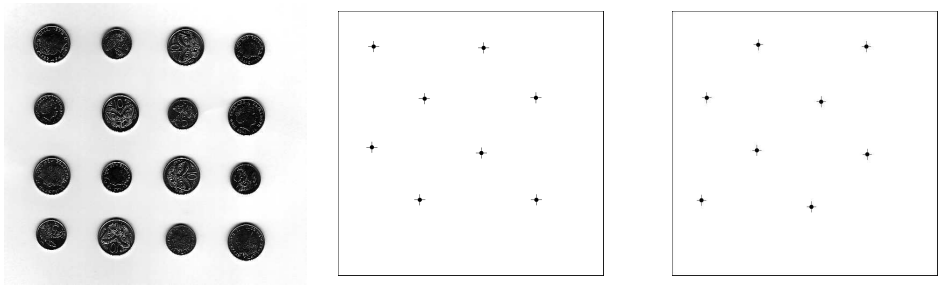
Feature Set Investigation

From these results in table 5.3, two important conclusions can be drawn regarding the effect of using different terminals (features) on program solutions. The first is that circular-based feature region terminal sets (IV and V) proved more effective as they both found solutions faster and produced better solutions. Both yielded results with no false alarms and solutions were found in under 20 generations, compared to the other square-based feature region terminal sets which took between 34 and 128 generations to find solutions produced some false alarms.

The second is that programs using terminal set I (5 rectilinnear quadrants) both evolved faster and achieved better results than terminal set II (central quadrant). This is important as terminal set II was constructed as a simplified or reduced feature set of terminal set I to test whether all 5 quadrants were needed as features, or if they could simply be reduced to just a central quadrant. The results show that the programs generated using only the single central quadrant were not able to produce ideal results in the similar number of averaged generation as the rest were. The reason for this is simple: the single quadrant was not sufficient or a powerful enough feature by itself to distinguish between the 2 classes of interest as it was too small to capture the most noticeable difference - the different sizes. In other words, where terminal set I provided the features to accomplish this, terminal set II could not. Therefore according to this detection problem, using only 1 central quadrant did not provide sufficient features to separate the different classes of interest, more were definitely required.

This conclusion contrasts the conclusion asserted regarding the effectiveness of terminal set II on the easy images. On that particular problem, programs evolved using this terminal set were found faster than those using terminal set I because of the reduction of the search space. This implies that for easy detection problems a simple terminal set is more effective, but for more difficult problems such as this one, a more extensive set of features are required.

Two example object detection maps are presented below using terminal set V. In figure 5.2, (a) corresponds the original image, (b) to class 1 (*10c*) and (c) to class 2 (*5c*), which show that an ideal detection performance was achieved.



(a) (b) (c)

Figure 5.2: Sample detection map using Terminal set V

Evolved Programs

The average program sizes for each of the different terminal sets gave similar patterns to the tables presenting the training times for each terminal set (table 5.3). Of the three programs evolved using terminal set IV, the averaged size was the smallest, only 36, while terminal set II the largest, with an averaged size of 179. Programs evolved using terminal set V produced the next smallest averaged size of 66, followed by terminal set I with 107, and terminal set III with 113.

These program sizes reflect the usefulness of the different features used in the terminal sets, where the smaller the programs, the more powerful the features. Since the circular-ring region based terminal set (IV) allowed for the smallest programs to be evolved in the shortest times, they must be the most powerful. Following closely are the next set of circular features, the whole-circular region features (terminal set V).

The square-based features produced programs about 2 or 3 times larger than the averaged size of terminal set IV and V, suggesting that for this particular problem, the square-based regions were not as powerful as the circular based region features.

Presented below is a program evolved using terminal set V (a) and one evolved using terminal set II (b). They both performed the same on the detection process, each producing a 100% DR and a 0% FAR, but program a was found considerably faster than program b. It is interesting that even though both achieved the detection performance, program a is 5 times smaller than program b.

Program a, size = 35

```
(d+ (d+ (d/ F6 (d- T (d/ F2 T))) (d- (d- T F7) (d/ T F5))) (d* (d/
(d- T T) (d- F3 (if F5 F1 T))) (d- F7 (d+ F8 (if T F1 T)))))
```

Program b, size = 183

```
(d/ (d- (d- (d+ (d* F3 (d* F3 T)) F4) (d- (d* (d* F2 T) (d- T F3))
(d* (d+ T F4) (d+ F1 T)))) (d+ (d+ (d* (d/ F2 T) (d+ T T)) (d/ (d-
T F4) (d/ T F4))) (if (if (d* T F2) (d+ F3 F1) (d* T F4)) (d- (d- T
F1) (d* F2 T)) (d- (d/ F3 T) (d/ T F4))))) (if (d* (d- (d+ (d* F1 T)
(if T F2 T)) (d/ (if F3 T F1) (d- F1 T))) (d- (if (d+ T F2) (d+ F2
T) (d* T F4)) (d* (d/ T F1) (d/ F2 T)))) (if (d- (d+ (d/ T F4) (d-
F1 T)) (d* (if F1 T F2) (d/ T F1))) (d+ (d/ (d- F4 T) (d- F4 F1))
(d/ (d- T F1) (d+ F2 T))) (d+ (d* (d+ F4 F4) (d* F1 T)) (d* (if F2 T
F3) (d/ F4 T)))) (d/ (d/ (d/ (d+ F1 T) (d* T F2)) (d/ (d+ T F1) (d*
F4 T))) (d+ (d+ (if T F1 T) (d- T F2)) (d+ (d+ T F3) (d/ F4 T))))))
```

5.2.2 Results for Object Classification Applied to Detection (OCAD)

Three programs for each terminal set were trained using the (OCAD) method. The following two subsections present a summary of the classification performances as well as how these programs fared when applied to detection.

Classification Training Results

Since classification allows for the opportunity to train on multiple images – to prevent overfitting – the training set used for this problem was increased to 3 images (compared to 1 using straight-forward detection). More specifically, each of the three images contained: 16 object cutouts corresponding to the centers of 16 objects of interest in an image (8 for each class) and some background cutouts (24). The reason there are three times as many `background` cutouts is that this class must include both pure background, and background combined with portions of an object of interest. This is to 'prepare' the classifier to evolve programs that can be applied to object detection too. The number of cutouts is calculated as follows:

$$[8(class1) + 8(class2) + 24(backg)] * 3(images) = 144 cutouts$$

Table 5.4 presents the training performance of each terminal set on the cutouts.

Terminal Sets	*Classification Accuracy*			
	class1	class2	background	**Total**
TermSet I	100	100	99.1	**99.7**
TermSet II	100	100	99.61	**99.53**
TermSet III	100	100	98.61	**99.53**
TermSet IV	100	100	99.1	**99.7**
TermSet V	100	100	99.33	**99.77**

Table 5.4: Training performance for classification on 10c and 5c coin cutouts

None of these programs achieved an overall classification accuracy of 100%. Although each class of interest was able to do this, the `background` class was not. This is due to

the selection of background cutouts that included large portions of true objects. The classifier could not correctly classify those cutouts as background since some included portions covering even more of the cutout than the background. These more difficult cutouts were chosen in the aim of 'preparing' the programs for the sweeping procedure used in detection.

The training times for this method were short, usually on the order of 2-10 minutes.

Detection Results

Table 5.5 shows the object detection results achieved by the OCAD method. The *best* evolved program from classification was applied to the test set in a sweeping window fashion.

Terminal Sets	*DR*			*FAR*		
	class1	class2	**Total**	class1	class2	**Total**
TermSet I	100	100	**100**	100	0	**50**
TermSet III	100	100	**100**	100	100	**100**
TermSet III	100	100	**100**	100	100	**100**
TermSet IV	100	100	**100**	100	0	**50**
TermSet V	100	100	**100**	50	0	**25**

Table 5.5: Classification-applied to detection results for 10c and 5c coins

Term Set V performed the best, only generating half the number of false alarms, on average, for class 1 (5 cents). Comparatively the rest of the terminal sets were worse, all generating either:

- All (100%) false alarms for class 1 *heads* (Term sets I, IV) – reporting all objects of class 2 as class 1 – and no false alarms for class 2 *tails*, or
- All false alarms for both classes (Term sets II, III) – reporting every single object of either class as a false alarm for the other.

It is important to note that the centers of the objects were classified correctly. It was the portion around the object that triggered a false alarm. This resulted in the unclustered centers producing a 'hollow' in the middle of an object-of-interest of the incorrect class in the output detection maps, as the centers did not generate any false alarms.

None of these results are ideal. Considering that the results generated using the SFD detection method on this problem were ideal, the OCAD method does not seem appropriate for object detection. However, the evolved populations that are a result of this method does prove useful when substituted for the initial population in the two-phase training method (see results below).

5.2.3 Results for Refining Object Classification For Detection (ROCD)

Since the straight-forward detection method was able to achieve ideal results on this problem, the aim of this method was only to possibly reduce the overall training times. Two evolved populations from the classification method were used in evaluating these results. The 2 populations correspond to the program evolutions using terminal sets I and IV. These two were chosen because they both had relatively long training times in the original method that could possibly be reduced. In each case, three separate programs were evolved using this 2-phase training method and the averaged results were results reported.

The effect of substituting an evolved population for the initial population did result in a slight improvement. Both training times were reduced compared to straight-forward detection, while both terminal sets maintained their 100% detection rate and 0% false alarm rate. The average number of hours taken to find the solution for the programs evolved using terminal set I was reduced from 26 to 16, while for programs evolved using terminal set IV this dropped from from 17 to 11 hours.

Evolved Programs

The averaged size of the *2-phase evolved* programs were also shorter compared to those programs evolved using straight-forward detection. The average program size for terminal set I dropped from 66 to 43. A more important observation was for the programs evolved using terminal set IV which dropped from 107 to 56. This is a simplification by a factor of approximately a half. This suggests that this 2-phase evolutionary process has an additional benefit apart from just reducing training times, it also favours the evolution of simpler programs than the straight detection method (both using the same fitness function). Presented below is an example of this shorted evolved program. It was generated using terminal set I, and is of size 56

```
(d- (d- (d/ (d/ T (if F3 F3 F4)) (d- F6 T)) (d/ (if (d* T F3)
F4 (if T F5 T)) (d+ (d/ F12 T) (d+ T F6)))) (d/ (d/ (if (d+
F9 T) (d+ F11 T) T) F4) (d+ (d/ (d* T F11) (d* F10 T)) (d/
(d* F11 T) (if T F12 T)))))
```

5.3 Task 3 — 5 Cent NZ Coin images

In approaching task 3, evolving programs using the first method, *SFD*, did not achieve perfect results. On every training attempt, there was always at least 2 false alarms that prevented the convergence of a solution. In most cases, the training was aborted after reaching the maximum number of generations allowed without finding the optimal one. The best programs fitness did not improve after about 80 generations, even if the maximum number of generations was increased.

For this reason the two other methods discussed earlier, *OCAD* and *ROCD* were also applied to this task in an attempt to improve the performance. These methods however, were also unable to find the perfect solution during training and testing, but the results do show an improvement.

5.3.1 Results using Straight Forward Detection (SFD)

Each evolved solution was applied to three unseen images and the averaged detection and false alarm rates were reported.

Terminal Sets	*DR*			*FAR*		
	tails	heads	**Total**	tails	heads	**Total**
TermSet I	100	100	**100**	97.5	27.5	**62.5**
TermSet II	100	100	**100**	100	30.7	**68.5**
TermSet III	100	100	**100**	87.5	12.5	**50**
TermSet IV	100	100	**100**	55.35	23.21	**39.28**
TermSet V	100	100	**100**	100	25	**62.5**

Table 5.6: Straight-forward detection results for 5c heads and 5c tails coin images

None of the terminal sets found the ideal solution. All five achieved a 100% detection rate for both classes but with some unwanted false alarms. Terminal set IV, the whole-region circular features achieved the best performance, producing the lowest (on average) false alarm rate (FAR), 39.28%, compared to the others which were all 50% or above. This FAR corresponds to roughly 4-5 false alarms for class $tails$ and 1-2 false alarms for class $heads$, from 8 object per class in each image. The worst performance was from terminal set II, the 4 rectilinnear features, producing a 68.5% FAR. These results suggest that just 4 rectilinnear features may not be sufficient, as the 12 rectilinnear features performed better, achieving a lower FAR (62.5 %). Terminal set I and V performed roughly the same both yielding the same FAR's, but worse than Terminal set III (50 %).

It is interesting to note that in each case, the false alarm rate for class $tails$ was always higher than the false alarm rate for class $heads$. This can be clearly seen from the graph (figure 5.3), which presents the overall false alarm rates for all the terminal sets, split into those generated by each class.

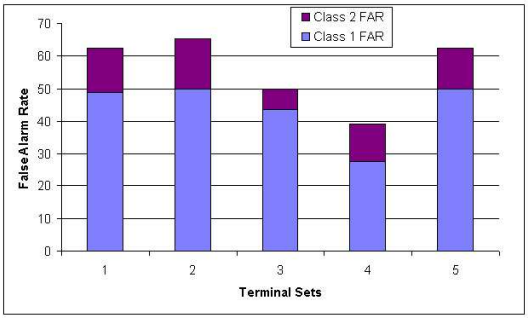


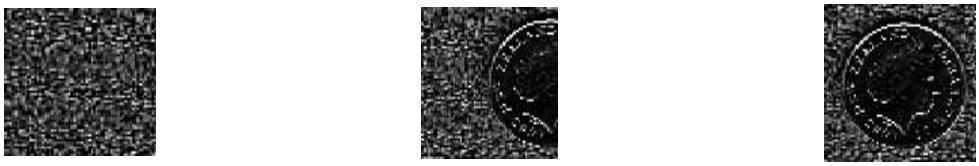
Figure 5.3: False alarm rates for all 5 terminal sets

An important conclusion can be drawn from the graph. All the terminal sets have roughly a similar proportion of false alarms for class 2 ($heads$) over class 1 ($tails$) even if their overall FAR vary, implying detecting objects for class 1 ($tails$) was more difficult than class 2 ($heads$).

Analysis and Discussion

From analysis on the centers produced by the detection system, it was observed that all the false alarms for class 1 ($tails$) were generated along the borders of some of the objects of class 2 ($heads$). That means that none of the *centers* of class 2 ($heads$) were reported as false alarms for class 1 - these were correctly classified as class 2 - it was only where the input

window was half over the object, a false alarm was generated. Figure 5.4 shows an example of this 'half-object-in-moving-window' problem.



(a) Class Background (b) False Alarm (c) Class 2

Figure 5.4: Half an object in the moving window generating a false alarms

From figures 5.4, (a),(b) and (c) correspond to input windows during the sweeping detection procedure. Figure (a) shows a moving input-window classifying the background preceding an object-of-interest correctly. Figure (b) shows the same moving input window containing half of object of class 2 (*heads*), which was incorrectly reported as a false alarm for class 1 (*tails*). Figure (c) correctly reports the same object, as seen in figure (b), as a detected object center for class 1 (*tails*). This illustrates one of the main problems the GP detection system encountered with this task: reporting a large proportion of false alarms.

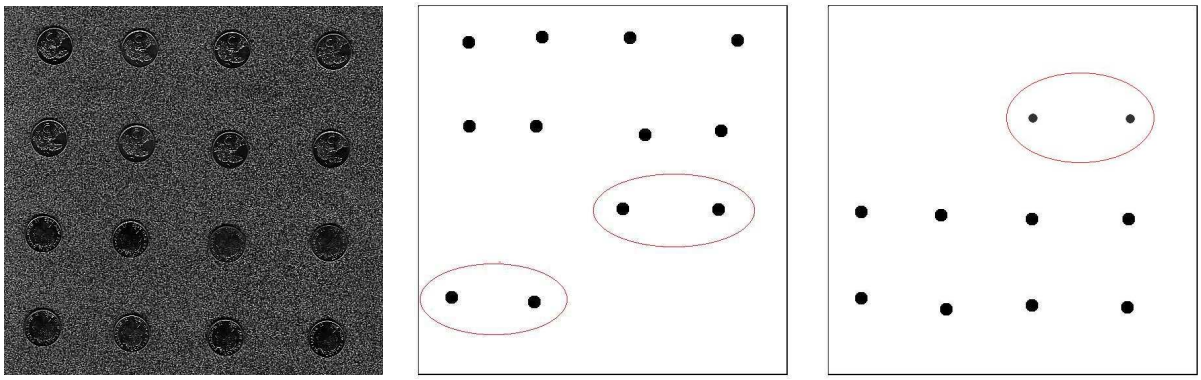
A major factor contributing to this problem is the pixel statistics extracted from the moving windows. For example, considering the case where the mean value of the pixels in the moving window over some `background` is less than threshold `T`, no object is reported (figure 5.4 (a)). After moving 72 pixels across (width of object), when the moving window is centered over an object of class 2, the mean will be twice than the threshold `mean > T+T` and an object of class 2 will be reported (figure 5.4 (c)). During this sweeping procedure, there will be some points where the mean value passes through `T < mean < T+T` generating a false alarm (figure 5.4 (b)), as the mean of class 1 lies between `T < mean < T+T`.

A good program would learn to separate these values with the appropriate use of the `if` function in the solution.

There are two other important factors contributing to the large number of false alarms. First, the number of TOLERANCE pixels allowed. If this number was larger, less false alarms would have been generated as the clustered centers would have extended to include them. The disadvantage in increasing this value is that the programs do not learn to separate these values. Therefore the lower it is, the more the programs learn to incorporate the difference into the learning phase. Second, the weighting of the constants in the fitness function. Recall that K1 (DR) was weighted far greater than K2 (FAR), thus favouring the selection of programs with a high DR first. If these constants were weighted more evenly, programs with a less FAR would have been favoured effectively lowering the overall FAR of the best program. The risk in that is that the DR could drop to lower that 100%.

Sample Object Detection Maps

Presented in figure 5.4 are object detection maps to illustrate the detection results. These maps were obtained by a good evolved program using terminal set IV. As we can see, all 16 objects-of-interest of both classes were detected (100% DR) but with 6 false alarms (37.5% FAR). In these figures, the small black dots represent a detected object center, while the larger red ellipses around a detected center show the occurrence of a false alarm. Note that these ellipses were manually added to the detection maps to show the false alarms, they were not output from the GP as the black dots were. (Note, this detection map was generated from the program presented below).



(a) (b) (c)

Figure 5.5: Detection map showing all objects found, but with some false alarms (circled)

From this we can see that the 6 false alarms were not split evenly between the two classes. Four were reported for class 1 (*heads*) (b) and two for class 2 (*tails*) (c), suggesting *heads* were harder to detect than tails. For both classes, the GP detection system successfully localized each of the objects, reporting the centers only a few pixels within range of the true centers. From this we can also see that the GP system successfully learned the difference between object and background as the only false alarms generated are other objects, not pure background.

Sample Programs

Presented below is an evolved program using *terminal set IV*. This program produced the object detection map above. The program size is 62, where the average size of all programs generated using this approach 64.

```
(d+ (d- (if (d+ (d- F3 T) (if T F2 T)) (d+ (d- F4 T) (d/ F4
T)) (if (d+ T F1) (d+ T F5) (d/ F5 F3))) (d* (d/ (d* F2 F2)
(if T F3 T)) (d/ (d/ F1 F3) (d/ F1 F2)))) (if (d* T (d- (if T
F3 T) F4)) (d+ T (d- (d/ T F5) T)) F6))
```

Note that this average program size is almost twice as large as the average size of those programs evolved using the same terminal set for the easier coins problem (10c vs 5c images), which was 35. This is mainly because this problem is more difficult to solve.

5.3.2 Results of the Object Classification Applied to Detection method (OCAD)

Classification Training Results

In a similar way to 'classification applied to detection on 10c and 5c Coin Images', the classifier trained on a number of cutouts. Since this is a more difficult problem to solve as both classes of interest (heads and tails) are of the same size and the background is highly cluttered, the number of cutouts was increased to:

$$[8(class1) + 8(class2) + 35(background)] \times 3(images) = 153 cutouts$$

Table 5.7 presents object classification results on the cutouts. For each terminal set, three runs were performed in the experiments and the averaged classification accuracy over the *three* program evolutions were reported.

Terminal Sets	*Classification Accuracy*			
	tails	heads	background	**Total**
TermSet I	100	100	98.32	**99.44**
TermSet II	100	100	97.17	**99.05**
TermSet III	100	100	97.81	**99.27**
TermSet IV	100	100	98.4	**99.46**
TermSet V	100	100	98.53	**99.51**

Table 5.7: Classification-applied to detection results for 5c heads and 5c tails coin cutouts

Similarly to the training performance for classification on the 10c and 5c coin problem, none of these programs achieved an overall classification accuracy of 100%. This is again due to the classifiers inability to correctly classify some of the more difficult background cutouts which included large portions of an object in the cutout.

Similarly to the training results for the OCAD method on task 2, the training times here were between 5 and 10 minutes.

Detection Results

The best program evolved from classification was directly applied to the large images in the test set in a sweeping window process. The object detection results are presented in table 5.8.

Terminal Sets	*DR*			*FAR*		
	tails	heads	**Total**	tails	heads	**Total**
TermSet I	100	100	**100**	100	20	**64.5**
TermSet II	100	100	**100**	100	50	**75**
TermSet III	100	100	**100**	100	45	**72.5**
TermSet IV	100	100	**100**	100	0	**50**
TermSet V	100	100	**100**	100	20	**60**

Table 5.8: Straight-forward detection results for 5c heads and 5c tails coin images

Like with the *easier coin problem* (section 5.2.2), the classification-trained programs applied to detection did not yield very good detection results. All programs kept up the 100% detection rate but most resulted in an increase in the false alarm rates (FAR) compared to straight-forward detection. Also like straight-forward detection, the majority of the FAR's was reported for class 1 (*tails*) , implying that the classifier was more successful in detecting class 2(*heads*) than class 1 (*tails*).

The most significant difference is that all the terminal sets produced a steady 100% FAR for class1 (*tails*) , whereas this rate for class 1 varied between 50% and 100% for straight-forward detection. A major contributing factor to this would be the 'half-object-in-moving-window' problem as discussed earlier (section 5.3.1), as the classifier trained on far less fitness cases (cutouts) than the straight detection system.

It is interesting however that from all 3 programs evolved using terminal set IV, none reported any false alarms for class 2 *tails* (0% FAR). This terminal set is the only one out of all five terminal sets to achieve this, and is also an improvement on the results using straight-forward detection, which had a class 2 FAR of 23.21%. This is a positive result as these solutions were generated using far less fitness cases and in a considerably shorter training

time than the *straight-forward* detection method. The total false alarm rate for this method was larger however (50%) compared to the *straight-forward* detection method (37.5%) because of the bad performance for class 1 *heads* (100% FAR).

Feature Set Analysis

In terms of terminal set performance, again the whole circular-region features (Term set IV) achieved the best results with the lowest overall FAR rate of 50%. The worst performance was also again from terminal set II, producing the highest FAR of 75%.

For a more intuitive view of how the terminal sets fared against each other, Figure 5.6 shows the false alarm rates produced by the 5 terminal sets, using both detection methods seen so far. The detection rates were not included in the graph as these remained the same (100%) for all terminal sets.

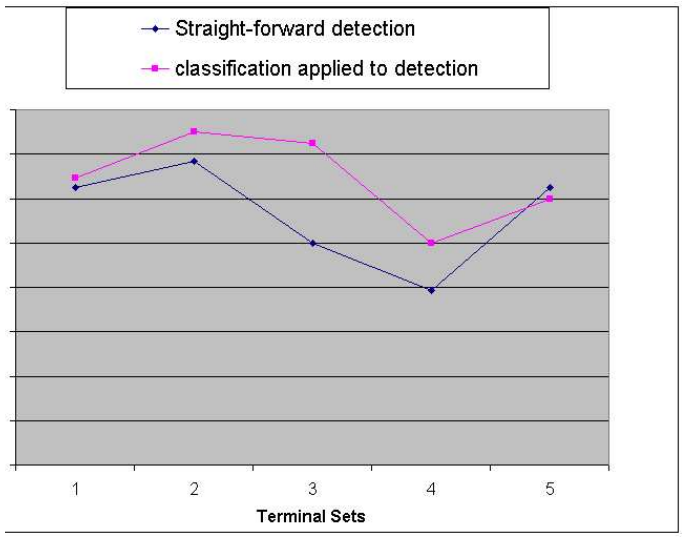
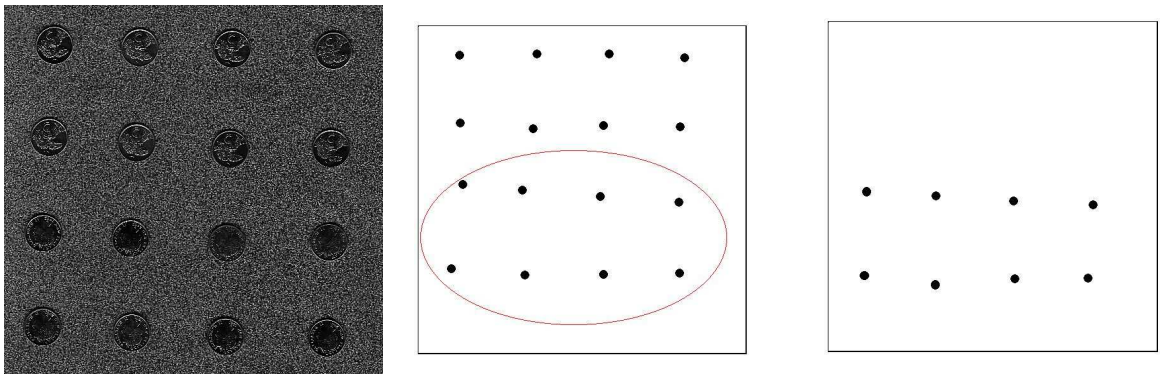


Figure 5.6: Comparison of of terminal set performance for both method seen so far

Since both methods show the performance of the circular-region terminal sets better than the square-region terminal sets, it can be concluded that for this particular task circular-based regions are a more powerful set of features.

Figure 5.7 shows sample object detection maps achieved by an evolved program using terminal set IV. As we can see, all 16 objects-of-interest of both classes were detected (100% DR) but with 8 false alarms all for class 1 *tails* (none for heads).



(a) (b) (c)

Figure 5.7: Detection map showing a overall 50% FAR (all false alarms for class tails

Sample Programs

The average sizes of programs evolved using different terminal sets were between 34 and 93. Those using terminal set IV were the lowest at 34, not far from those using terminal set V which was 42. The square-based feature set programs were larger, with both terminal sets I and III averaging at 62 and terminal set II the highest at 93. Program a presented below was generated using terminal set IV, which produced the detection maps shown in figure 5.6.

Program a, size = 39

```
(d- (d- (if (d- F5 F6) (d/ T F2) (d+ F7 F6)) (d/ (d* F2 F3) (d- T
F5))) (if (d- (if T F4 T) (d+ T F5)) (d* (d* F1 F3) (d/ F2 F3)) (d*
(d/ T F6) (d/ F3 F5))))
```

5.3.3 Results for Refining Object Classification for Detection (ROCD)

This approach was tested using the best terminal set and evolved population from the OCAD method. That is, **TermSet IV** using population corresponding to **program** a (from program presented in OCAD method). The reason only one terminal set and population was tested in this section was to investigate the effects of the secondary-training phase on a population, not to compare how this method evaluates on different terminal sets.

Three programs were evolved separately, using the same population and each evaluated on the test set. Table 5.9 presents the average detection performance of the ROCD method using terminal set IV.

Terminal Set	DR			FAR		
	tails	heads	**Total**	tails	heads	**Total**
TermSet IV	100	100	**100**	55	0	**27.5**

Table 5.9: Detection results for the ROCD method on 5c coin images

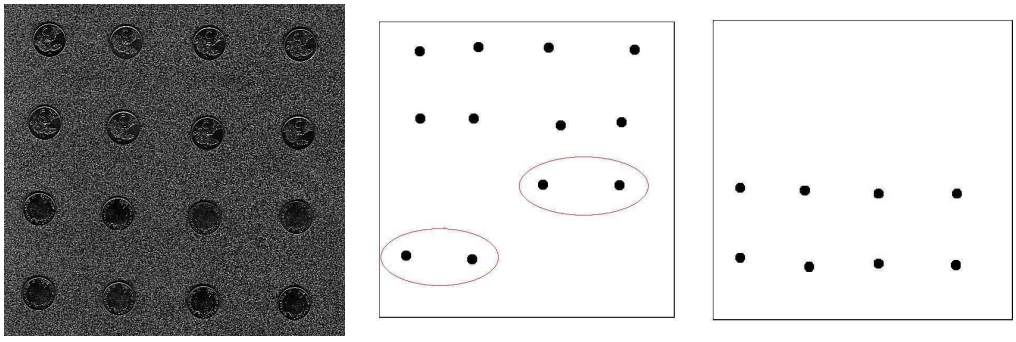
This result is better than the results from the first two approaches, showing the lowest total false alarm rate of 27.5% . The 100% detection rate was also maintained, as well as the 0 false alarm rate (FAR) for class2 (*heads*). The difference is that there are less false alarms for class 1 (55%) compared to 100% using classification. This asserts that the secondary training

phase did have a positive effect in reducing the number of unwanted false alarms from the best evolved program trained using the classification-approach.

These results were obtained after a maximum number of 100 generations was reached in the secondary training phase, and similarly to the previous two methods, the ideal solution of a 0% FAR was not achieved. Extending the maximum number of generations did not make a difference on any of the three training runs.

The secondary training process was similar to the straight-detection training process in terms of training time, as every program needed to be applied to the entire image on every generation. The 'most-evolved' program however was achieved faster, in terms of generations, than in the first method (where 'most evolved' implies a program which can not be improved upon any longer via evolution). Recall that for straight forward detection, the most-evolved program was reached around 60 generations of evolution compared to 30 in this case. This difference can be attributed to the already partially evolved initial population used in training compared to the randomly generated initial population used in straight forward detection.

The 27.5% FAR was calculated from the following performances by the three evolved programs on the test set: programs 1 and 3 achieved a 26.25% FAR (reporting mainly four false alarms out of 16 true objects per image) and program 2 a 28.75% FAR (reporting mainly five false alarms). The best object detection map from a test image based on the first evolved program is presented below (the original image is the same as the one initially presented in straight-forward detection).



(a) (b) (c)

Figure 5.8: Detection map for ROCD method for task 3.

These images show that the secondary training phase helped to reduce the original 50% FAR as seen from the best detection map using the classification (section 5.3.2), to 25%, but did not completely eliminate it. Further investigations need to be carried out on whether false alarms can be eliminated if a better set of GP parameters or classification strategy is used.

Evolved Programs

Presented below are the one of the three genetic programs evolved using this method. It corresponds to the program generating the detection maps from figure 5.8.

```
Program size = 72
(d- (d/ (d- T F3) (d/ F3 T)) (d+ (d/ (if T T F3) (d+ (d+ (if F4 (d+
T F5) T) (if (d/ T T) (d/ T F1) (d* F2 T))) T)) (if (d+ (if F1 (d*
(d- F3 F2) (if F3 T T)) T) (if (d/ (d- T F3) (d/ F3 T)) (d- T T) (d*
T (d+ F2 T)))) F2 (d* (d* (d* T F3) (d/ (d+ T T) F2)) T))))
```

The average size of programs evolved using this ROCD method was 66. This average is roughly the same as from the straight-forward detection method (62), both larger than the classification applied to detection method.

5.3.4 Performance Comparison of the Three Methods

Table 5.10 shows a summary of the best results from each detection method, which were achieved using terminals set IV.

Detection Method	DR			FAR		
	tails	heads	**Total**	tails	heads	**Total**
Straight-forward	100	100	**100**	55.35	23.21	**39.28**
Classification	100	100	**100**	100	0	**50**
Refining classification	100	100	**100**	55	0	**27.5**

Table 5.10: Performance comparison of the three approaches on 5c heads and 5c tails images

As discussed earlier, each achieve a 100% detection rate both all classes, which is very important in any detection system. The only difference in terms of performance results between these methods is the false alarm rates. The graph below presents a more intuitive view on how these varied for the 3 methods.

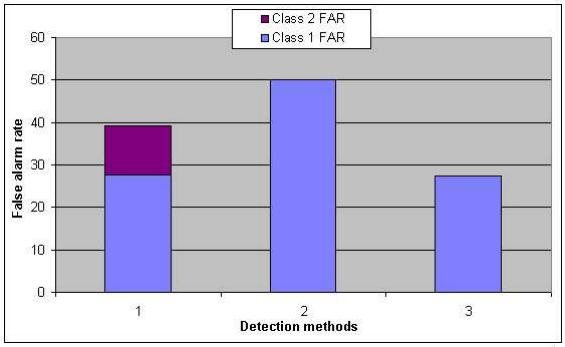


Figure 5.9: False alarm rates for 3 detection methods using terminal set IV

It becomes easy to see that the secondary-training phase served to eliminate the false alarm rate for class 2 entirely, dropping the total FAR to 27.5%. Straight-forward detection was unable to do this for class 2 and therefore has a higher total FAR of 39.28%. The secondary-training phase was able to this because of the initial population that was used as input from the classifier (method 2). We can see that even though the classifier produced the highest FAR from the 3 methods, it was successfully able to learn the difference between class 2 and class 1 – something the straight-forward detection system was not able to do – and this was the advantage. The secondary-training phase simply refined these relatively small programs to achieve the better performance.

5.4 Task 4 — 10c and 5c Heads and Tails NZ Coin Images

In task 4, there were 16 objects of interest in the images, four belonging to each of the four distinct classes (5c tails, 5c heads, 10c tails, 10c heads).

Similarly to the results from task 3, none of the detection methods produced evolved programs able to achieve an ideal performance on this task. On every training attempt, there was always at some number of false alarms which prevented the convergence of a solution. In most cases, the training was aborted after reaching the maximum number of generations allowed without finding the optimal one. The best programs fitness did not improve either after about 80 generations, even if the maximum number of generations was increased.

5.4.1 Results using Straight Forward Detection (SFD)

Three programs were evolved for every terminal set and each applied to test set. The results presented in table 5.11 are the averaged detection and false alarm rates of each terminal set over the test set.

Terminal	*DR*	*FAR*				
Sets	**Total**	tails05	heads05	tails10	heads10	**Total**
TermSet I	**100**	200	175	100	250	**181.25**
TermSet II	**100**	200	200	100	300	**225**
TermSet III	**100**	150	150	125	275	**175**
TermSet IV	**100**	250	125	100	275	**187.5**
TermSet V	**100**	275	175	0	100	**137.5**

Table 5.11: Results using straight forward detection on task 4

Like the previous problem (task 3) none of these results are ideal (or even close to it). All the programs were able to detect all the objects for each class easily (100% DR) but also a high overhead of a large false alarm rate (FAR).

A few conclusions can be inferred in the FAR's over the terminal sets and four classes. The most distinguishing conclusion is that for each terminal set, the FAR was greater than 100%. That means, more objects were reported as false alarms than there were true objects in the images. These FAR's are the highest produced thus far for any of the detection problems, suggesting that this task was the most difficult for the GP system to solve.

Another conclusion is that in most cases, with the exception of terminal set V, the highest individual FAR was for class 4 (*heads10*). The asserts that nearly all the terminal sets found class *10c heads* the most difficult to solve. This could either be because class 4(*10c heads*) simply 'looks' like every other class according to the detectors, or the more likely cause that the class 4 range of detector outputs was difficult to separate according to the classification map.

If this is the case, then the high FAR can be attributed to the similar 'half object' in the moving window problem mentioned earlier (section 5.3.1.). Since the class 4 range in the classification map is above all the other values, the program (detector) output must pass through all values *lower* than this when the input window moves from background (output less than 0) to the center of an object of class 4. This allows a higher chance of false alarms to be generated while the program output values move from 0 up to the range `3*T < output < 4*T`. A 'good program' would have learned to separate this with an `if` function or/and

powerful features, which is the most likely cause for terminal set V producing the low overall FAR. This suggests that the function or terminal sets are not powerful enough, or the classification strategy is inappropriate for this five class problem.

Feature Set Comparison

The terminal sets had varying success in solving this problem. Figure 5.10 presents a graph showing the false alarm rates separated by class for each of the terminal sets.

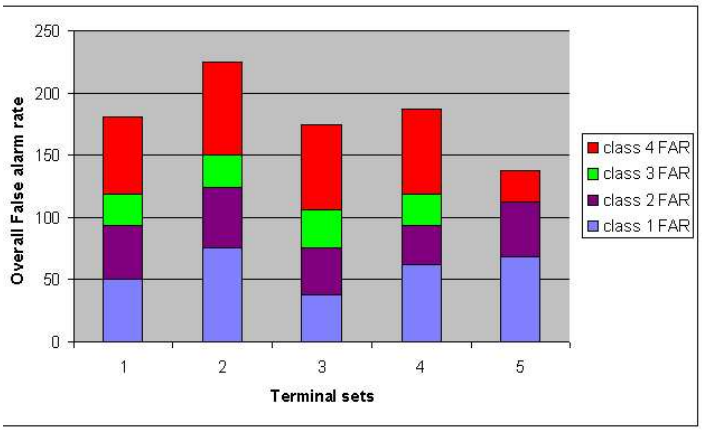
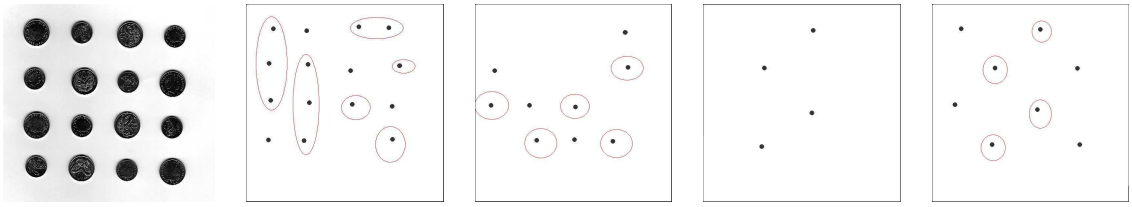


Figure 5.10: False Alarm rates for all the terminal sets

As can be seen from the figure, terminal set V produced the lowest total FAR, and was the only class to achieve a 0% FAR for one class – class 3 (*10c tails*). Once more, the highest FAR was produced by terminal set II, the smallest set of features. Unlike the previous problem, the square and rectilinnear region features (terminal sets I and III) achieved a lower FAR than the whole-circular region features (terminal sets IV).

Sample Detection maps

Figure 5.11 presents sample object detection maps achieved by a good evolved program using terminal set V. In the images the small black dots represent a detected object center, while the larger red ellipses around a detected center shows the occurrence of a false alarm.



(a) (b) (c) (d) (e)

Figure 5.11: Detection map showing all objects detected but with many false alarms (light red ellipses)

From these detection maps we can see that all 16 objects-of-interest for all four classes were detected (100% DR) but with a total of 20 false alarms (125% FAR). Note that class 3

(*10c heads* (figure 5.11c) was the only class with no false alarms. Also present in this specific detection map is the detectors ability to separate the 10c coins (figures 5.11c and d) better than the 5c coins (figures$a$ and b). This is attributed to the obvious size difference between these classes.

Sample Programs

Presented below is an evolved program generated using terminal set V (generated the detection maps from figure 5.11). The average size of these programs was 86. Note that this average size is the largest seen so far for any of the four tasks.

Program size = 83:

```
(d- (d/ (if (d- (d- (d+ F3 F4) (d- T F1)) (d/ (if T F5 F4)
(d- F3 T))) (d* (d- F5 (if F5 F4 T)) (d- (d+ F3 F4) (d- T
F1))) (d* (d- F2 F5) (d+ (d/ F7 F3) (d+ F7 F3)))) (d- (d+ (if
(if F1 F4 F8) (d- T F7) T) T) F5)) (d- T (d/ (if (d- F3 T)
(d/ (d+ F4 T) (if T F8 F6)) (d+ (d- T T) T)) (d- (d/ (d* T T)
F7) F5))))
```

5.4.2 Results for Object Classification Applied To Detection (OCAD)

This method and the method immediately proceeding (ROCD method) uses the best terminal set and population from the SFD method. That corresponds to terminal set V. Details on the training results were omitted as they were very similar to the performances from this method on two previous tasks. The number of cutouts was increased to a total of 264, where the majority of these include a large number of background cutouts with portions of true-objects contained within the cutouts in the goal of reducing the false alarms. The cutouts number of cutouts were chosen according to:

$$[4(class1) + 4(class2) + 4(class3) + 4(class4) + 76(backgre)] * 3(images) = 264 cutouts$$

All three program solutions trained to a classification accuracy of between 92-95%. The remaining incorrect classifications were due to the increased number of 'difficult to classify' background cutouts.

Presented below are the detection results of the best evolved program trained from above on the test set.

Terminal	*DR*	*FAR*				
Sets	**Total**	tails05	heads05	tails10	heads10	**Total**
TermSet V	**100**	300	300	300	300	**300**

Table 5.12: Results for object classification *applied to* detection on task 4

These results are not good. Although all the objects of interest were found (100% DR), there was a large FAR. This large FAR (300%) asserts that the classification-evolved program regarded every single object as a false alarms for every class. This means that these programs were unsuccessful when applied to detection as they failed to sufficiently learn the difference between the four distinct classes of objects.

5.4.3 Results for Refining Object Classification for Detection (ROCD)

For this method, three programs were evolved using the population from the OCAD method. Table 5.13 presents the averaged detection results of the *2-phase evolved* programs on the test set.

Terminal	*DR*	*FAR*				
Sets	**Total**	tails05	heads05	tails10	heads10	**Total**
TermSet V	**100**	300	300	0	0	**150**

Table 5.13: Results for refining object classification for detection on task 4

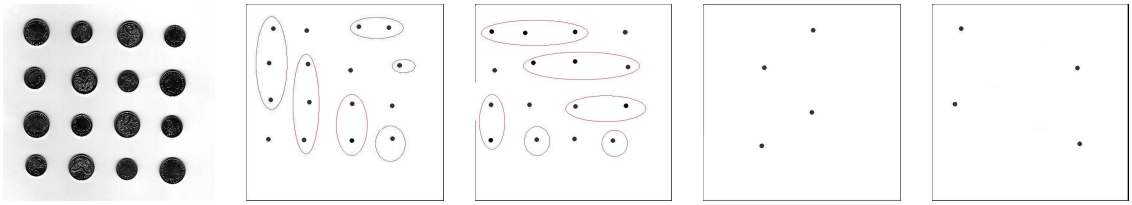
These results are interesting. On the one hand, the total FAR is slightly higher than the FAR achieved using the straight-forward detection (137.5%), suggesting that this method was not useful in reducing the large number of FAR's generated on this problem. This was due to both classes 1 and 2 (5c heads and tails) producing a 300% FAR, reporting every object in the image as a detected center for that particular class.

On the other hand, the FAR is both 0% for classes 3 and 4 ((*10c tails* and *10c heads*). This result is good as not even SFD method was able to achieve it. This asserts that unlike the case for class 1 (*5c tails*) and class 2 (*5c heads*), the detectors successfully learnt the difference between classes 3 and classes 4 evaluated against to the other two classes.

The reason for the higher FAR's for class 1 and 2 was that it based on the classification-trained programs. But the question as to how the other 2 classes (3 and 4) FAR's were reduced is difficult to answer. After some analysis on the programs, it was noticed that while evolving the programs to sort the detectors output for the 10c coins classes (3 and 4) – which the GP system found easier – the parts of the program that dealt with the 5c coin classes (1 and 2) were either disrupted or ignored altogether being simply too difficult to separate. This problem was further complicated by the large sizes of detectors evolved for this task. The chance of disrupting useful sub-trees increases with larger programs, as these sub-trees become increasingly reliant on other sub-trees in the program. This is know as *synergy* - 'the interaction of two or more agents or forces so that their combined effect is greater than the sum of their individual effects'. This problem suggests that the GP system needs to be improved (see 'Future Work, section 6.2).

It can be initially concluded therefore that the task of separating the program outputs into 4 different regions in the classification map proved almost too difficult for **this** GP system to solve.

Presented in figure 5.12 is sample object detection maps (produced from the program below). It shows there are many false alarms occurring only for class 1 and 2, but perfect results for class 3 and 4.



(a) (b) (c) (d) (e)

Figure 5.12: Detection map for 2-phase training

Sample Programs

The sizes of programs generated using this method, as mentioned above, are large. The averaged size was 216. Presented below was the shortest of these three programs (size 202).

```
(d- (d/ (if (d/ (if (d+ F3 T) (d+ T F5) (d+ F6 F2)) (d- (d-
F2 T) (d- F1 F3))) (d* (d/ (d- F1 T) (d- F1 F5)) (d- (d* F2
T) (d* T F1))) (d/ (if (if T F7 T) (d- T F6) (d* T T)) (d-
(d- T T) (d+ F1 F4)))) (if (d* (d/ (d/ T F4) (d* F5 T)) (d/
(d* F6 T) (d* T F7))) (d+ F1 (if (d+ T F4) (d- F3 T) (d+
F7 T))) (d* (d+ (d* T F4) (d* T F6)) (d* (d* T F2) (d* F8
F3))))) (if (if (d/ (if (d+ F5 F1) (d+ T T) (d+ F6 T)) (d+
(if F7 T F8) (d- F1 T))) (d* (if (d+ T F2) (if F3 T F5) (d/
T F4)) (d* (d* T F5) (d* F6 T))) (d/ (if F5 (d- T F8) (d* T
F1)) (d- (d- F2 T) (d- T F3)))) (d/ (d* (d/ (d- F5 T) (d- T
F5)) (d+ (if T F6 T) (d/ T F7))) (d* (d* (d/ F6 T) (if T F1
T)) (d+ (d+ F2 T) F2))) (d- F1 (d+ (d/ (d* F2 T) (d- T F3))
(d- (d* T F4) (d* F5 T))))))
```

5.5 Summary

This section contains three parts: first is a summary of the fitness function comparison, followed by a summary showing of the detection results using the three different methods on each of the tasks, and finally a summary of the feature set performances on each of the four tasks.

5.5.1 Fitness Functions with Program Size

Table 5.14 presents a summary of the average program size of program detectors evolved using both fitness functions compared in this project (equation 4.3 and equation 4.4 in section 4.4.1). These fitness functions were only compared on the first task because this task had the lowest overall training times from all the four tasks, allowing for a more comprehensive comparison.

Fitness function	average size of evolved program detectors
Fitness(DR,FAR)	62
Fitness(DR,FAR, FAA, ProgSize)	24

Table 5.14: Summary of fitness function comparison on the first task

Table 5.14 shows that the average size of program detectors evolved using the multi-objective fitness function with program size were almost three times smaller than those program detectors evolved using the simple fitness function. Consequently, these shorter programs were easier to interpret, contained less redundant subtrees and were evolved faster because less time was needed (in training) to evaluating them.

5.5.2 Detection Results Summary Using the Different Methods

Table 5.15 presents a summary of the **best** detection performances for each of the three methods developed in this project (namely SFD, OCAD and ROCD) on the four tasks. The detec-

tion results correspond the detection and false alarm rates achieved by the best terminal set on the task. Note that *n/a* means *not applicable* (no results were generated using the method concerned).

Tasks	*SFD*		*OCAD*		*ROCD*	
	DR	FAR	DR	FAR	DR	FAR
Task 1	100	0	n/a	n/a	n/a	n/a
Task 2	100	0	100	25	100	0
Task 3	100	39.28	100	50	100	27.5
Task 4	100	137.5	100	300	100	150

Table 5.15: Summary of detection performance of each method on the four tasks

From table 5.15, the following conclusions can be inferred:

- All three methods were able to successfully find all objects of interest for all the tasks (100% DR).
- Overall, the GP system found the four-class object detection problem (task 4) the most difficult to solve, followed by the noisy-background problem (task 3). The computer-generated and easy coin images proved the easiest (first two tasks) to solve.
- The OCAD method was unsuccessful when applied to detection as it always generated a larger FAR than the other two methods. This method did however serve its purpose in supplying the 2-phase method (ROCD) with an evolved population produced in relatively short training times.
- The ROCD method was able to improve the detection performance from the OCAD method by achieving a lower number of false alarms.
- In most cases, the two-phase method, ROCD, achieved the best performances and the shortest overall training times.

5.5.3 Feature Set Performance Summary on the Tasks

Table 5.16 shows a summary of which terminal set out of the five being investigated in this project (namely I, II, III, IV and V) was responsible for achieving the best performance on each of the four tasks. Included is a brief justification as to how the terminal set contributed to solving the tasks.

Tasks	Best Terminal Set	Justification
Task 1	Terminal set II	Fastest solution convergence
Task 2	Terminal set IV	Fastest solution convergence & smallest program sizes
Task 3	Terminal set IV	Best detection performance & smallest program sizes
Task 4	Terminal set V	Best detection performance & smallest program sizes

Table 5.16: Summary of feature set performance on the four tasks

Table 5.16 suggests that terminal sets IV (six circular region features) and V (eight circular-ring region features) were generally more effective and better suited for the object detection

tasks than the others, with the exception of terminal set II on task 1. The main reasons concentric circular features outperformed the other square-based features are two-fold. First, terminal sets IV and V produced the lowest number of false alarms in the two difficult tasks (3 and 4) compared to the programs evolved using the other sets. Second, these features were powerful enough to allow shorter programs to be evolved over longer ones seen in programs using the other terminal sets.

Terminal set II (four recti-linnear features) on task 1 is the only exception to the trend of circular features performing better than square-region feature because it is the terminal set with the least number of features. On the easiest task where all the other terminal sets were also able to achieve ideal detection results, terminal set II allowed for the fastest convergence of a solution due to the reduced search space (four features). This performance was not consistent however as the difficulty level of the tasks increased. In those cases, it was noted that the worst results were from terminal set II due to the lack of powerful enough features.

Chapter 6

Conclusions

This chapter presents the main conclusions from this project, followed by a future work discussion.

6.1 Main Conclusions

The goal of this project was to develop and test three domain independent approaches to multi-class object detection problems of increasing difficulty using genetic programming. Five terminal sets were developed using domain independent, low-level pixel statistics as well as a new fitness function constraint. This fitness constraint factored the programs size into program evolution according to Ockhams law, where the smaller the solution structure, the better.

During the development and experiments in this project, the following characteristics can be concluded as important to successfully solving the multi-class object detection problem using genetic programming:

1. In terms of the three different approaches developed in this project,

   - Using the straight-forward detection method, the GP system was able to solve the 4 tasks reporting a 100% detection rate for each case. Perfect detection results (no false alarms) were achieved on the computer-generated images (task 1) and easy 10c *vs* 5c coin images (task 2). The while the noisy 5c heads *vs* 5c tails coin images (task 3) and the extended *four-class* problem (task 4) proved more difficult to solve as there were always some unwanted false alarms reported by the program detectors. The training times for this method were long, and further research is needed as to whether the false alarms can be eliminated if a better set of GP parameters or classification strategy are used.
   - Although the classification method was able to achieve a 100% detection rate for each of the four tasks. While the training times were very short in comparison to the first approach, this method always caused a very large number of false alarms to be reported. This method therefore, was not sufficient when applied straight to detection as the evolved programs were not robust enough to deal with the sweeping-window procedure used in detection. However, it can provide a good baseline as a starting point for the secondary-learning phase in the ROCD method. Further research is need on a better selection of objects-cutouts used for training.
   - The secondary training phase was able to reduce the training times for all four tasks. In addition, for most tasks, this method also achieved a reduction in the

number of false alarms reported. The ROCD method always improved or *refined* the detection performance from the OCAD method.

- For most tasks, the ROCD method achieved the best detection performances. For the task where this was not achievable (task 4), it did serve to illustrate that the inefficiency in the high number of false alarms was not due to the method of approach, but rather to the fixed-threshold classification strategy which clearly needs to be developed, or a more powerful function operator to deal with a larger number of input classes.

2. The circular-based feature sets proved more effective and powerful than the square-based feature sets, as programs evolved using those features achieved better results on the tasks (generating the least number of false alarms), were smaller in size and were found faster during the evolutionary process. Similarly, programs using the feature set containing all 5 rectilinear quadrants proved more effective than the those using single quadrant feature set. This asserts that the usefulness of the additional four bordering quadrants in extracting more powerful features than just the single quadrant itself, which proved inefficient on the more difficult tasks.

3. The two extra constraints to the fitness function, *false alarm area* and the *program size*, were beneficial to the evolutionary process. It served to both reflect smaller changes to programs and favour the evolution of smaller programs. An important advantage of using a multiple objective fitness function is also that it makes the search space more continuous as opposed to being 'terraced and flat', because even more information about the evolved genetic programs such as tiny improvements can be reflected via their fitness. The *program size* constraint was especially useful as it succeeded in evolving shorter and easier-to-interpret program structures, which not only increased the comprehensibility of the evolved genetic programs but also reduced the time taken to evaluate these programs in the training phase.

4. Overall, the performance of this GP approach deteriorated as the difficultly level of the problems/tasks increased. Task 1 and 2 were solved with ideal results while task 3 and 4 both reported an increasing number of false alarms respectively.

Although these detection results are promising, there are some limitations to the use of GP for object detection. Some such limitations as discovered in this project are the long training times, the inability to reduce the number of false alarms reported as the difficulty levels for the problems increased, and the dependence of many different free parameters (to be set) required for evolutionary run where the choice can effect the final detection results. The long training times are a serious limitation as it not only hindered evaluating the GP systems performance on different images and problems, but also experimenting with different evolutionary parameters. Such parameters include the crossover and mutation operator rates, the constant weight values in the fitness functions, the threshold `T` used in the classification map, different function and terminal sets, the optimal population size and *minimal* maximum program depth allowed.

In conclusion, the GP approach discussed in this project was successful in automatically finding multiple classes of objects of interest contained within a range of images. The approach has the potential to expand on a wider range of object detection problems and tasks. Certain issues however must be addressed if this approach to automatic object detection is to 'evolve' along with other state of the art computer vision systems.

6.2 Discussion and Future work

This section briefly introduces several interesting research questions as discovered during the process of this project. Due to time constraints these issues were not developed, but would indeed be a good starting point for further research in this field. Some of these include:

1. Improving the classification method using a dynamic classification map or an $n-way$ binary classification approach.
2. A *Bayesian* probabilistic approach to using genetic programming as a classifier.
3. Improving the OCAD and ROCD methods.
4. Exploring the effects of using different evolutionary parameters, such as crossover vs mutation rates, and different weight-value constants in the fitness function.
5. More effective function and terminal sets.
6. Wider range of images.

6.2.1 Improving the Classification Method

A *dynamically-allocated* classification map

The classification-map approach for the classification phase of the detection process needs to be improved. This was apparent from high number of false alarms reported as true objects in the two difficult detection tasks (the two-class coin problem on a noisy background for task 3 and the four-class coin problemf or task 4). The high number of false alarms for the two-class coin problem illustrated that having a fixed, linnear classification map with intervals separated by user-defined constant `T` for every class is ineffective. This was because program outputs close to, but not actually class 2, were mapped as class 1 (not correctly as 'background') simply because the range for class 1 outputs on the classification map immediately preceded those for class 2, thus generating a false alarm.

A better method will be to adjust the intervals in the classification map (`T` values) so that they allow program outputs to cluster according to the different classes of objects of interest in the problem. These clusters could be dynamically adjusted or fixed in size, where the focus then shifts onto separating the program outputs into different *slots* and each slot is mapped to a different class. This removes the constraints of having to separate all the program outputs of one class according to a user different threshold `T`.

For example, requiring all class one program outputs below `T` and all class two program outputs above `T` is difficult for two reasons. Firstly, what is the optimal `T` value. Secondly, how can we know whether the program outouts for class one should proceed the program outputs for class two. If the order of these are reversed, will the classification be simpler? This problems complexity increases with the number of classes in a given task.

***n-way* Binary Classification Approach**

Similarly, the GP system produced a large number of false alarms for the four-class detection problem, suggesting that this problem caused the most difficulty. This is because the GP system had to not only separate objects from background, but also separate each of the four distinct classes from each other: class1 *vs* class2 *vs* class3 *vs* class4 *vs* background.

A different approach would transform this four-class problem into a n-way binary classification and detection problem, where n is the number of classes of interest. That is, treat each of the four classes as an *object vs background* problem, where everything else (all other classes) is background. This simplifies the problem, as if each each class is only evaluated against the background, the GP system does not have to learn to separate all four distinct classes.

This could be implemented in two ways. The first would be to use n different classification maps (fixed or dynamically allocated), one for each class. At every sweeping-window location, the genetic program is evaluated four times on each of the classification maps. This method however still places tough constraints on the genetic programs as one program would still have to solve all n classes. The second method is more time and resource consuming, but would place less constraints on the genetic programs. It involves evolving n different populations simultaneously and using the same evolutionary parameters, function and terminal set, and fitness function, except each population solves one of the binary classification problems.

6.2.2 A *Bayesian* probabilistic approach to using genetic programming as a classifier

Could we change the method of classification done by the genetic programming system to a more *probabilistic-based* one? Currently, the most fit evolved individual is responsible for all the predictions according to its program output. This method does not take any into consideration the prediction of the next most fit individual, or the one after that even though they too might have reasonably good fitnesses.

A probabilistic-based approach would take into account the predication of *all* the individuals in an evolved population not just the most fit. Each prediction is weighted by the fitness of the individual responsible for it. Further more, this could be done for each class, and a value obtained corresponding to the probability a given cutout or sweeping-window location is class x, given all the programs output.

This is advantageous as if the most fit individual has learned to separate all classes of interest except for one, while all the other less fit programs were able to correctly learn this classes output but are useless at accurately predicting the output of the other classes, a probabilistic approach will have the best of both worlds. In the context of the high false alarm rate problem, it could help in eliminating or reducing this for difficult problems.

Formally, this can be justified according to the standard *Bayesian Rule for Prediction*. This rule, which states that the probability of D^{new} is:

$$P(D^{new}|\mathbf{D}) = \sum_{h} P(D^{new}|\mathbf{D}, h).P(h|\mathbf{D})$$

where D^{new} is the new data to be classified, $\mathbf{D}$ is the current data and h is a *hypothesis* for the structure of the underlying data.

This formula can applied to GP and classification as follows:

$$P(class_i|population) = \frac{\sum_{p=1..PopSize}(Program\_output_p = i).Program\_fitness_p}{\sum_{p=1..PopSize} Program\_fitness_p}$$

where

- $P(class_i|population)$ is the probability that a moving-window location is classified as class i given the evolved GP population,

- $(Program\_output_p = i)$ will return 1 if program p predicts that this current window location is of class i, and 0 if not (the actual prediction - $P(D^{new}|D,h)$)
- $Program\_fitness_p$ is the fitness of program p (belief in the strength of the hypothesis - $(P(h|D))$
- Required to normalize the probability.

6.2.3 Improving the OCAD and ROCD Methods

The object classification applied to detection method (OCAD) did not yield programs robust enough to use in the sweeping window approach, as they always generated some false alarms. This method however was promising as it did significantly reduce the training times. There could be ways to improve this method such as explicitly merging this method with the *refining 2-phase* training method, or employing a more rigorous selection of object cutouts to use in training. Merging this method with the refining 2-phase training method would involve splitting the evolution process into two stages. The first, say for x generations, would evolve a population using cutouts as fitness cases. The second, say for y generations (or until the optimal solution is found), would then continue evolving the population using the sweeping-window approach. A more rigorous selection of object-cutouts involves a more extensive and 'hand-picked' set of cutouts being chosen as the fitness cases that that deal especially with the 'half-object-in-moving-window' problem encountered responsible for the high number of false alarms being reported.

6.2.4 GP Parameter Setting

Many different evolutionary parameters can effect the overall performance of evolved programs such as population size, the maximum allowed depth of programs, the number of TOLERANCE pixels allowed for clustering detected centers, etc. Presented below are two of the most important ones as encountered in this investigation.

Crossover and Mutation Rates

There is considerable debate in the research field questioning at what approximate points in program evolution does mutation become more constructive than crossover [16]. It is generally accepted that crossover is very useful early in program evolution, but gets increasingly harmful toward the later stages as they have a higher chance of disrupting potentially useful and larger subtrees. Is there some stage in program evolution where these rates can be dynamically altered depending on the fitnesses of the programs? For example, in the first 10 generations apply a 70% crossover rate and a 28% mutation rate (where the remaining 2% is left for the elitism operator), after x generation have passed change this ratio to 49% and 49%, and finally in the last y generations of evolution, apply a 28% crossover rate and a 70% mutation rate.

Weight Values for the Fitness Function

Recall that the fitness function used was

$$fitness(DR, FAR, FAA, ProgSize) = K1*(1-DR)+K2*FAR+K3*FAA+K4*ProgSize$$

where K1, K2, K3, K4 were all user defined constants that reflected the weight of that particular component for overall the fitness function. In this investigation, K1 was the weighted

the highest, followed by K2, K3 and lastly K4 which had the lowest weighting. This was to reflect that the detection rate was the most important, followed by the false alarm rates and lastly the program size. This ensured that programs first find all objects of interest, then attempt to reduce the false alarms. A disadvantage was that for some of the difficult problems, this caused the false alarm rates to remain high. The relative weighting of these constants could be set to dynamically update according to program evolution, in a similar way the crossover and mutation rates can be changed above. For example, changing the value of the false alarm rates weight after some time to be the highest weight value (over the DR weight value) could lead to a reduction in the overall false alarm rate reported by a program. The best time to change these values is the crucial question as if it is done too early in the process when all the objects of interest are not detected (less that 100% DR), chances are they will not be detected later.

6.2.5 More Effective Function and Terminal Sets

A factor in the GP systems inability to solve the difficult detection problems with ideal results could lie in the function and terminal sets not being powerful or sufficient enough. Future work could focus on evaluating a richer and more extensive set of functions which could include the max/min pixel value of a region, mathematical and trigonometric functions such as $sqrt$, exp or sin, or even the $sigmoid$ function which is highly utilized in neural networks. More extensive terminal sets can also be explored such as feature regions formed from a combination of circular and square regions.

6.2.6 Wider Range of Images

This project used real-world generated images where the objects of interest were both uniform ND symmetric in shape (circular). This made the tasks relatively easy. It would be interesting to explore the range of images the GP detection system using domain independent pixel statistics could solve. For example, consider the weather prediction and hurricane warning system application mentioned earlier (section 1). In this case the hurricane objects are not symmetric in shape nor are they of the same size.

Bibliography

[1] *Computer Vision and Image Processing: A Practical Approach using CVIPtools*. Prentice Hall, PTR, 1999.

[2] *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[3] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. Genetic programming: An introduction on the automatic evolution of computer programs abd its applications. *Morgan Kaufmann Publishers, Heidelberg: Dpunkt-verlag*, ISBN: 1-55860-510-X, 1998.

[4] C.L. Blake and C.J. Merz. Uci repository of machine learning databases, 1998.

[5] Steven Brumby, James Thieler, Simon Perkins, John J. Szymanski, and Neal R. Harvey. A genetic programming approach to extracting features from remotely sensed imagery. Technical report, Los Alamos National Laboratory, 1998.

[6] J. Eggermont, A.E. Eiben, and J.I. van Hemert. Comparing genetic programming variants for data classification, 2001.

[7] John Hagedorn and Judith Devaney. A genetic programming system with a procedural representation, 2001.

[8] Christopher Harris and Bernard Buxton. Low-level edge detection using genetic programming: performance, specificity and application to real-world signals. Technical report, 1997.

[9] Neal R. Harvey, Steven P. Brumby, Simon Perkins, James Thieler, John J. Szymanski, Jeffery J. Bloch, Reid B. Porter, Mark Galassi, and A. Cody Young. Image feature extraction: Genie vs conventional supervised classification techniques. *IEEE Transactions on Geoscience and Remote Sensing*, 2001.

[10] Daniel Howard, Simon C. Roberts, and Richard Brankin. Target detection in sar imagery by genetic programming. *Advances in Engineering Software 30, pg 303-311*, 1999.

[11] A. Jain, J. Mao, and K. Mohuiddin. Artificial neural networks: A tutorial. *IEEE Computer*, 29(3):31-44, March 1996.

[12] John R Koza. *Genetic Programming: On the Programming of computers by means of natural selection*. MIT Press, 1992.

[13] Mykola Pechenizkiy, Seppo Puuronen, and Alexey Tsymbal. Feature extraction for classification in knowledge discovery systems. Technical report, Computer Science Department The University of Dublin Trinity College Ireland, July 2003.

[14] Mark Prichard. Genetic programming for multiclass object detection, 2002.

[15] Walter Alden Tackett and K Govinda Char. Genetic programming applied to image discrimination. *Handbook of evolutionary computation, Oxford University Press*, 1997.

[16] Mengjie Zhang. *A domain Independent Approach to 2D Object Detection Based On the Neural and Genetic Paradigms*. PhD thesis, Dept of Computer Science, RMIT University, Melbourne, 2000.

[17] Mengjie Zhang, Peter Andreae, and Mark Prichard. Pixel statistics and false alarm area in genetic programming for object detection. *Proceeding of the European Conference on Genetic Programming for Image Analysis and Signal Processing, Lecture Notes in Computer Science. To appear*, Vol. 2611., 2002.

[18] Mengjie Zhang, Victor Cielielski, and Peter Andreae. A domain independant approach to multiclass object detection using genetic programming. *Journal of Applied Signal Processing*, 2002.

[19] Mengjie Zhang and Victor Ciesielski. Genetic programming for multiple class object detection. *In Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI'99) , published in Norman Foo (Ed.): Lecture Notes in Artificial Intelligence, Springer-Verlag Berlin Heidelberg, pages 180-192. Sydney, Australia*, LNAI Volume 1747, December 1999.

Appendix A

Programs and Packages used in this Report

A.1 Strongly-Typed Genetic Programming Package (RMITGP)

The Genetic Programming package used and modified for the purposes of this research project was written by Dylan Mawhinney from the *School of Computer Science and Information Technology, RMIT University, Australia* in 2002. It is a strongly-typed GP package written in `c++` for research. Being strongly-typed means that the package is capable of using user-written and problem-specific functions and terminals of different types in the evolutionary engine. The following classes were written and added to the package to transform it into something that could solve the multi-class detection problem:

- A new **fitness class**: responsible for reading feature vector, class name pairs from a patterns file, defining an appropriate fitness function and classification strategy, and applying these to a population of genetic programs. Two separate fitness classes were written: the first for object detection (methods 1 and 3) and the second for object classification (method 2) .

- A new **function class**: Defined the 4 arithmetic operators and a conditional `if` operator.

- A new **terminal class**: Defined the terminals, where 1 terminal is represented by a floating point number corresponding to 1 image feature from the patterns file, and a constant threshold integer value `T`.

- A new **testing class**: Given an evolved program, a patterns file containing feature vectors corresponding to a test image, and the appropriate parameters used in training such as `T`, this class is responsible for applying this program to the unseen image to obtain the testing detection results. As output, the clustered centers of all detected objects were written to a file.

- Modifying the **main class**: When specified, this class allowed a population of programs to be read from a file and set as the current population to be evolved.

A.2 Supplementary Programs written for Feature Extraction and the GP-Output

A.2.1 Feature Extraction for the Patterns File

A number of programs where written (using c), responsible for extracting pixel statistical features from an image and converting them into features vectors in a patterns file. The first 5 programs (defining each of the 5 different local region features) accepted as input three arguments: an image in *pgm* format, a text file containing the center locations of all objects of interest and their class names, and the size of the sweeping input-window. The output was all the feature vectors corresponding to the all locations of the sweeping input-window (which could then be piped to a file). Similarly, 5 programs (for the 5 different local regions) were also written to extract features from the testing images. The only difference was that none produced and information regarding the classes or center locations of any objects of interest present in the image.

A.2.2 GP detected centers output

A program was written to display the detected centers output by the GP package in an image form. The program accepted as input a detected-centers file and produced an image containing black crosses over the locations that represented a detected center for visualisation purposes.