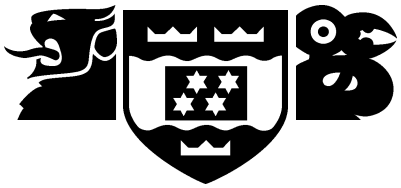
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



Computer Science

PO Box 600
Wellington
New Zealand

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

Improving Training Performance of Genetic Programming for Object Detection

Malcolm Lett

Supervisor: Dr. Mengjie Zhang

October 23, 2004

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

Abstract

Object detection has become an important research topic within computer science. Databases of images need to be searched, security images and speed camera images need image processing to search for various information, and there is an increased desire that computers should be able to recognise human faces and determine who they are. Genetic programming has been used for these sorts of tasks with varying success, however object detection is still a difficult task and can require long training times. This project investigates the task of finding the accurate positions of objects. From this investigation, two new fitness functions are devised which are competitive with existing methods in terms of detection rate, false alarm rate and time to evolve the solution programs when applied to data sets of increasing difficulty. Also produced from this investigation are guidelines for the types of data which should be trained on for object detection.

Acknowledgements

I would like to acknowledge first and foremost my supervisor, Dr. Mengjie Zhang, for introducing me to Genetic Programming, for being my teacher in both Genetic Programming and the joys and pitfuls of academic research, and for providing superb and helpful feedback on my writing and my ideas. And I'm sure that the constant deadlines pushed me to produce more than I may have done otherwise.

I would also like to thank Yun Zhang for discussions, suggestions, and criticisms throughout the year, and especially during the last days before deadline.

Lastly I would like to thank everyone else who has had any involvement in my experiences this year. Especially those in the Memphis room who have made things so much more interesting, and helped out many times during the year, not the least of which was L. Brook Powell who had the uncanny ability to solve my bugs even when I could barely explain my programs.

Contents

Figures

List of Tables

Chapter 1

Introduction

Object detection is the task of processing an image to both localise a particular object or objects and to then classify each object found. The ability to detect multiple classes of objects is important with the increasing number of images stored in databases, and with the increasing desire for computers to perform the tasks of humans when examining images for various information. Examples of tasks include the following:

1. Finding number plates in images of cars.
2. Examining ripeness/quality of fruit in orchards.
3. Reading scanned images of text (computer printed or hand written).
4. Face recognition.

Genetic Programming (GP) is a relatively recent addition in the field of artificial intelligence. It has been applied, among others, to the problem of classification and detection of objects within images. The most successful work in object classification and detection has been done in single class or binary class problems. Some prior work has been done on the issue of multiple class object localisation and classification and the various issues are being gradually resolved.

Detection and classification of New Zealand 5 and 10 cent coins in [2, 13, and others] has been done with good results (around 80-95% accuracy) for simple images of 3 or less classes, however results have degraded when more classes are to be detected and for images with noisy backgrounds which make them harder to classify. The results of localisation and classification have been varied so far. For simple images, with little to no noise and up to three classes, good results have been achieved. More recent work [15] has achieved close to 100% classification accuracy on even difficult images with 5 classes (heads and tails of 5 and 10 cent coins, plus background).

1.1 Issues & Motivation

Given an image there may be multiple objects present and these objects may be of different classes. The Localisation stage must determine the position of each object of all classes being looked for. These localised objects must then be classified. There has been research in both these fields but more work needs to be done to improve the efficiency and effectiveness (accuracy) of these localisations and classifications.

GP is used to evolve "programs" which solve a given problem. The evolution of these programs is governed by a "fitness function" which measures the "fitness" of an evolved

program to solve the problem. Constructing this function is done by hand and tends to be very difficult to construct well. Some existing work [11, 2] have added extra features as their need became known. However the most common methods have "free parameters" which have no defined value, nor obvious way to work out their value. Only rough tips and guidelines can be provided because they do not represent the real situation well.

Some specific issues in GP applied to object localisation which have not been satisfactorily addressed are:

1. Constructing a fitness function to be applied to the evolved programs is very difficult. For localisation, fitness functions must consider many different situations and these are hard to handle well with existing methods. If the training data contains errors, a fitness function which is not tolerant enough of errors will cause the effectiveness of trained localisers to be encumbered.

2. A trained localiser tends to find many possible positions for objects. These positions must be grouped together ("clustered") using some "cluster algorithm". The centres of these groups are taken as the position of the found objects. Calculation of the fitness of programs during evolution traditionally requires these "clusters" to be calculated, however this adds a very expensive processing overhead which slows evolutionary performance.

3. Selection of training data examples can affect the effectiveness (usefulness) of an evolved localiser program. Some work has focused on hand picking a small number of important training examples combined with randomly selecting a certain number of other examples[11, 2]. This hand picking of examples is cumbersome at best and may lead to errors in the generated programs due to poor selection of training examples. Problems with complex objects require greater number of training examples than for simple cases, increasing the time it takes to train an effective localiser. Some guides to what training examples are best to train with are needed.

This project will investigate and attempt to address these issues.

1.2 Research Goals

The goal of this project is to achieve better evolutionary performance and greater accuracy of detections by using better measures of program fitness, and by discovering rules for the selection of training examples which lead to efficient training of programs for the task of locating objects of interest.

The specific goals of this project are to:

1. Devise better methods for computing the effectiveness (fitness) of localisation program, which consider the complexities of the localisation problem. Determine what characteristics are desirable and incorporate these into the new methods.

2. Develop a method of improving training performance by eliminating the need for clustering during evolution.

3. Examine the effects of training data preparation on the evolutionary progress and effectiveness of programs produced, and discover rules for producing optimal selections of training examples for localisation.

1.3 Contributions

Two new fitness functions for the task of object localisation are introduced. These fitness functions are shown to perform extremely well when compared to an existing method.

The relative importance of different types of training data are discovered. This affects training performance as well as localisation effectiveness as training/evolutionary systems can train with more directed and useful training data than may otherwise be able to.

A paper entitled "New Fitness Functions in Genetic Programming for Object Detection" was accepted for publication in the Image and Vision Computing New Zealand Conference 2004 (IVCNZ'04), in Akaroa, New Zealand.

1.4 Structure of Document

This chapter has served has a brief introduction into the problems of Genetic Programming for classification and detection, and particularly considering localisation or detection. The remainder of this document describes the background theories and work in detail and presents two main new pieces of work: new fitness functions for localisation and a novel analysis of training data, along with analysis of the new methods. The structure is as follows.

Chapter 2 contains background information into Object Detection, the Genetic Programming paradigm, and how Genetic Programming can be applied to object detection. Chapter 3 describes the data sets with which our methods are tested. Chapter 4 introduces the theory and reasoning behind the development of two new fitness functions for object localisation and provides analysis comparing the new fitness functions to an existing method. Chapter 5 provides a detailed motive and method for examining certain characteristics of the training data and then devises an experiment and shows results to determine if good heuristic rules can be determined about the selection of training data. Finally, chapter 6 summarises the findings from the fitness functions and the training data analysis.

Some additional related information can be found in appendices A.1 to A.3.

Chapter 2

Background

This chapter serves as a brief introduction into some basic concepts of machine learning concepts, followed by a more detailed examination of the concepts within the Genetic Programming paradigm. Object Detection in Computer Vision is then discussed, with specific emphasis on the issues and performance measures relating to this report. Mention is then made to some specific issues in using GP for object detection. Finally, a brief literature review follows.

2.1 Overview of Machine Learning

Traditionally, programs have been written by people to perform tasks that change in pre-predicted ways. The programmers have had the insight necessary to handle the different situations. The programmers have also known how to deal with handling the data within their specific domain. If this has not been the case, the programmers have experimented until they have devised rules which model the environment in which their programs must work. Increasingly, the software being written is required to meet needs requiring much more complicated data processing. The data being processed is now so complex that often humans cannot easily discover good rules to describe the environment.

From a different direction, there is also the desire that machines should be able to adapt to their world through learning about it. There should be some way for a system, machine, or software program, to learn what its environment is, and then react in necessary ways to achieve the desired result. This is at a runtime level. At a design level, there is also the desire that software should be produced automatically, particularly when the task is too complicated for a human programmer to devise the rules that they should be incorporating into their software. Programmers wish to make their system learn how to perform the task. Once the task is learned, the program is then published as a working unit.

How does a computer learn?

This is a seemingly daunting task, and it has received much interest over many years. A lot of research has been done using one broad and seemingly basic method: a possible solution or set of solutions is *generated* and then *tested* for its ability (or *effectiveness*) in solving the problem. These solutions are initially generated by some random means. You can imagine selecting a few solutions from the set of all possible solutions, and then testing their fitness. The fitness of a solution is usually determined by trying it against the problem and seeing how well it performs. The next stage in this process is to determine what to do with the solution or solutions generated so far. Most systems take the existing solution or solutions and the information on how well they performed, and attempt to improve the solution(s).

Determining how to represent a problem is difficult, as many programmers know, and

determining how to represent and generate solutions, and then to measure the effectiveness of these solutions and to improve upon them, is extremely difficult and varies widely depending on the problem domain and the choice of the programmer. Broadly speaking, learning systems can be divided into three categories:

supervised There are 'correct' outputs which are already know for all inputs during training. Training data is often referred to as being 'labelled'.

unsupervised There are lots of inputs, but 'labelled' output data is rare or completely absent. Machine learning programs tend to look for arbitrary patterns relating entirely to the input values, such as in clustering algorithms.

reinforcement The inputs are not labelled with 'correct' outputs, however the system is occasionally told whether it has achieved the right or wrong answer. This 'reinforcement' is often delayed in time and the possible reason for the correct action or mistaken action cannot be directly deduced.

Some of the more common systems in use are: Neural Networks (NNs), Genetic Algorithms (GAs), Genetic Programming (GP), Decision Trees, and Bayesian Networks. These are typically used for either supervised or reinforcement learning.

Artificial Neural Networks (often abbreviated to Neural Networks) were inspired by the way that the human and animal brain function through the use of neurons which, at a basic level, transmit data to its outputs based on the value of the inputs and some learned behaviour. The learned behaviour maps certain inputs to a desired output by way of setting some computational parameters within each neuron.

Genetic Algorithms are a classic example of how difficult it can be to represent the problems and solutions. GAs require the solutions to be encoded into a string of values (typically binary values) like a chromosome would represent information. However this poses the difficult task of decoding the solution back into useful information, and also of how to represent the problem in order that such solutions can be generated.

Decision Trees and Bayesian Networks are best suited to inputs with discrete values. Decision trees are used as a classification system by dividing the set of possible outcomes by each of the inputs. Decision Trees are represented as a tree structure with each node representing a question, and the edges from the node representing each of the possible answers. Each answer could also represent a range of possible values for the input values. The learning system attempts to find the most broad and shallow tree by checking each possible input value and comparing how well it separates the known examples into separate and preferably unique subsets. The shallower the tree, the less inputs need to be examined to achieve an outcome. The broader the tree, the more expressive it is.

2.2 Genetic Programming

Genetic programming is an exciting method for evolving computer programs to solve complicated tasks. GP was originally devised in 1990 by Hugo de Garis[6], and since then John Koza has become the main proponent of its use, in [8] and later works.

GP is based on the Darwinian theory of evolution, where the fittest of a population survive but the unfit die. In GP, a *population* of programs is initially randomly created, each successive generation goes through a process of killing off the unfit programs, retaining the original programs, and possibly changing the population through evolutionary mutations.

GP is used as a supervised learning mechanism, using a training set of labelled examples, and some sort of *fitness function* is used to measure the fitness of programs produced against

this training set. This 'fitness' is used in determining when to terminate the learning system. The learning system is called the *evolutionary engine*.

2.2.1 Program Representation

Genetic Programming enables solutions to be represented as programs. These solution programs in GP are represented as variable sized tree structures. The leaf nodes are the inputs to the program, called *terminals*. The internal nodes form the functional body of the programs, they are formed from a function set that has been selected to be relevant to the problem domain. The internal nodes are called the *functions*. The root node produces a single output value, called the *result*.

Figure 2.1 shows a typical program produced by GP. The final output represents $(xyz - 1.34x^2)$. The programs produced are normally thought of as LISP style expressions (where the function is specified first, then its arguments, separated by spaces). The program in the figure represents the LISP S-expression `(- (* x (* y z)) (* 1.34 (* x x)))`.

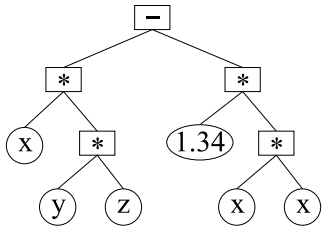


Figure 2.1: Program representation in GP

Function Set

Typically the functions all return single values or the same type. Often floating point numbers are used as the return type, however boolean operators, or functions returning integer numbers can be used also. Under Strongly Typed GP (STGP)[9], the functions are able to return values of different types. This poses many difficulties regarding the selection of the function set, the initial generation of programs, the mechanics of the genetic operators, and the selection of a good fitness function.

There is no limitation on the number of input arguments a function can accept, although typically each function would accept its own fixed number of arguments. The arguments can be terminal values or the return values of other functions.

Typical functions used include:

- basic algebraic operations: $\{+, -, \times, \%\}$
  where $\%$ represents *protected* division, whereby divide-by-zero situations are handled by returning the value 0 (zero),
- trigonometric operations: {sin, cos, exp, log},
- boolean operations: AND, OR, NOT, XOR,
- boolean tests: {IF bool-expr(a) THEN b ELSE c, IF a <0 THEN b ELSE c}
- other domain specific operations, such as commands for action.

Other, more complicated, programming structures, such as loops are not usually incorporated within the function set. Their use poses a great deal of complication to the evolutionary process and fitness measures.

Feature Terminal Set

The terminal set is comprised of some set of *features* extracted from the input domain, and optionally some constant values. The constant values are usually created as randomly generated numbers, but may also include other values such as 0.0 (zero) and other domain specific constants.

The set of features vary greatly depending on the problem domain and the requirements imposed by the designers.

Closure and Sufficiency

A useful pair of feature and function sets require two qualities, that of: *functional closure* and *sufficiency*.

Closure is a mathematical term saying that applying an operation on two inputs of some type, will result in a value of the same type. This extends to requiring that operators used in the function set will always result in legal values. Problems such as divide-by-zero situations leading to not-a-number (NaN) results must be dealt with by forcing these situations to some standardised answer, or exclude the operation from the function set.

Sufficiency requires that the function and terminal sets are sufficient to form a valid solution.

2.2.2 Operators

The programs in GP are "enhanced" by methods derived from biology. Programs are selected to undergo crossover, mutation, and reproduction at some specified rates. Their use is intended to provide improvements to existing solutions, to discard bad sections of existing solutions, and to ensure that the best solutions are not lost entirely. These operations happen on a random basis, to randomly selected programs, and consequently they have no inherent ability to produce better programs. Rather, crossover and mutation have an equal possibility of producing bad programs as good programs. The fitness function is then used to discard the bad programs and keep the good ones.

Crossover

Crossover takes two existing programs and produces two new programs. The old programs are replaced by these two new programs. The process involves taking a section from the first program and switching this with a section from the second program, as shown in figure 2.2. The "crossover points" select some sub-tree from each program. These points are chosen at random over the nodes within each of the full programs.

Mutation

Mutation is used to introduce new information into the population. Existing programs are mutated by randomly selecting a "mutation point". The sub-tree of nodes at this point is removed and replaced by a newly randomly generated sub-tree as shown in figure 2.3.

Reproduction

Also referred to as *elitism*, *reproduction* of programs is done by simply copying a fit program from one evolution to another. This is used to ensure the overall best fitness of the population to never decrease. So long as the current best fit program is never lost, the overall

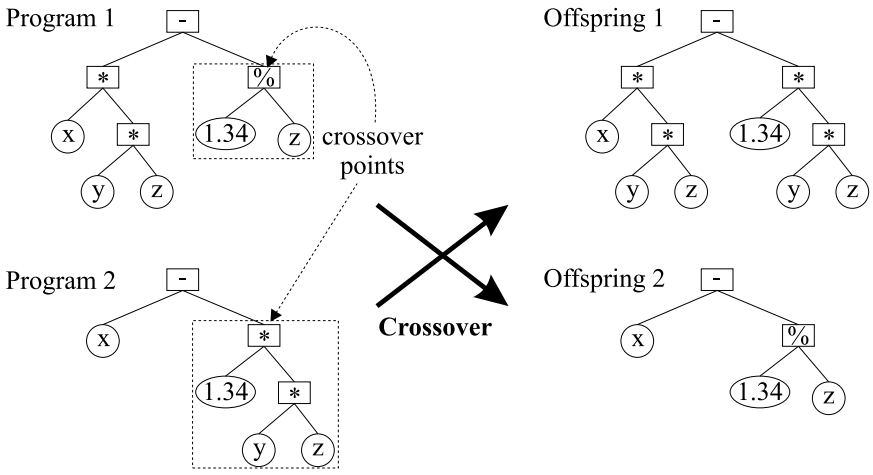


Figure 2.2: Example of crossover

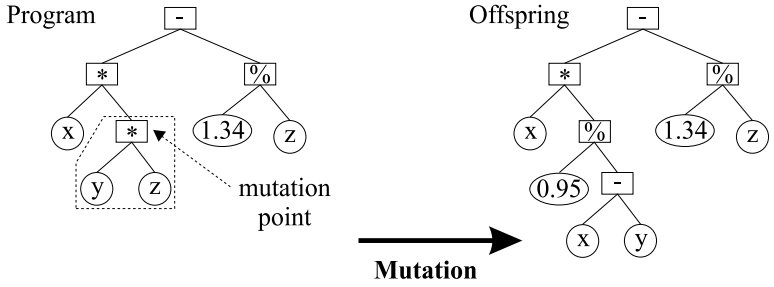


Figure 2.3: Example of mutation

population's best fitness will remain monotonically increasing (always equal or greater than previous value).

2.2.3 Fitness Function

The fitness function is some sort of function which measures how good a program is when applied to the *training set*. The training set consists of many examples (ranging from 100 to 100 thousand or more depending on difficulty of the problem domain and availability of examples), each of which have been pre-labelled with their desired output. For a classification problem, a typical fitness function is simply the "accuracy" of the program output. Calculated by counting how many examples the program produces the correct output for, and comparing this to the total number of examples the program is tested on. Another example is using RMS (root-mean-square) error for regression problems.

2.2.4 Evolutionary Engine

Whole populations of programs are created during evolutions in a run of GP. Each program is tested by applying it to some *training set* of examples. The test set examples have been pre-labelled with their correct output values and the fitness of the programs produced are measured by comparing their outputs with the required outputs. This is called *supervised learning*.

The engine for evolving programs in GP follows the steps shown in figure 2.4. The population is initially generated by randomly selecting the tree nodes from the function and

terminal sets. Each individual program within the population is tested against the training data and its fitness measured. The evolutionary engine is terminated when perfect fitness has been achieved or a predefined number of generations has been reached. When the evolutionary process is terminated, the best program produced so far is returned as the final output.

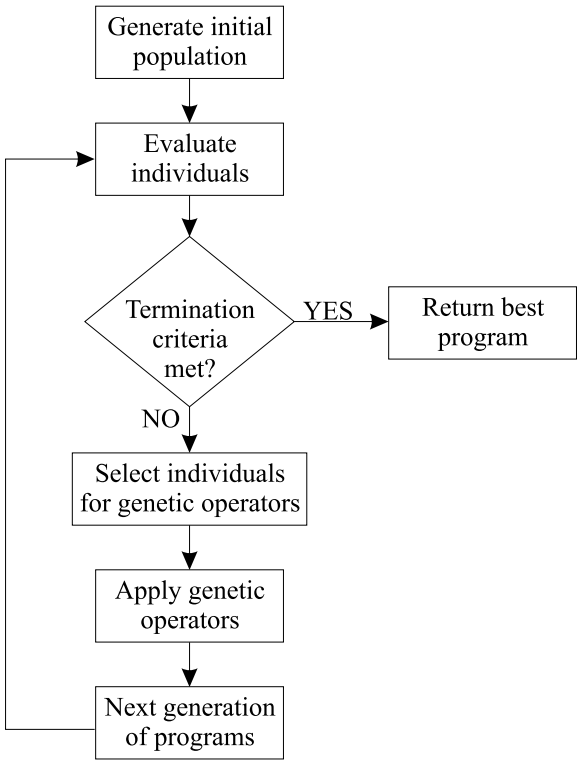


Figure 2.4: GP Evolutionary Engine

2.2.5 Termination and Validation Sets

Typically the GP evolutionary process will continue until one of two conditions are met:

1. The best fitness has reached ideal fitness. For example, 100% accuracy.
2. The maximum number of generations has been reached. This servers as a way of limiting training time if the programs never reach ideal fitness or wouldn't reach ideal fitness without a very long training time.

Using only a single set of training examples tends to cause *over fitting*, where the solution found works very well on the training data, but not very well on any other data it is later applied to. To solve this problem, the evolution is extended to use a second set of examples, called a *validation set*. The programs are applied to the validation set either after each test on the testing set, or at some regular interval of generations. The validation set isn't itself used to pick fit programs for evolution, rather it is used only to measure whether the programs are becoming over fit.

Figure 2.5 shows the idealised curves for the training set fitness and validation set fitness. As the training set fitness becomes increasingly better, the validation set fitness will initially get better also. However, at some point, the programs become *over fit*, being highly trained

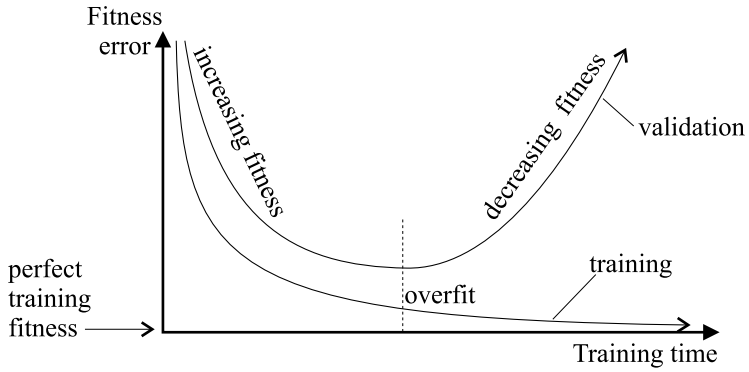


Figure 2.5: Train and Validation fitness curves

for the training set but not trained on the validation set, and consequently the fitness when applied to the validation set becomes worse. Evolution should be terminated at this point.

The system used within this project is very similar to this process, with the slight variation that the evolution is not terminated until the maximum number of generations or ideal fitness is reached. However, the validation set is applied after each generation and the generation with the highest validation set fitness is used as the best generation. The most fit program from the best generation is then applied to a third example data set, the *test set* and its fitness when applied to the test set, is used as the final measure of accuracy of the program produced. This ensures that the programs are not over fitted to either of the training set or validation set, and better measures the solution's generalisation. If the solution is not general enough, it will misinterpret entirely new examples of the same kind of data.

2.2.6 Summary of GP Parameters

Function set	A set of operations on terminal set values, typically something like $\{+, -, \times, \%\}$
Terminal set	The set of possible input values to the programs, varies greatly from domain to domain, and may contain constant values which are randomly generated during program generation and mutation.
Elitism rate	Percentage of programs selected for reproduction.
Mutation rate	Percentage of programs selected for mutation.
Crossover rate	Percentage of programs selected for cross-over.
Population size	Number of programs initially generated before evolution begins (typically large, around 500 programs)
Maximum program depth	Maximum size of programs allowed during evolution (typically around 7 to 15)
Minimum program depth	Minimum size of programs allowed during evolution (typically around 3 to 6)
Number of generations	Maximum number of generations attempted before evolution is terminated. (typically around 50 for easy programs, up to 200 or more for harder program)
Termination criteria	Method for deciding when to terminate, varies depending whether only only a training set is used, or if a validation set of examples is used also.
Training set	Example data set used to select fit programs for evolution.
Validation set	Example data set used to determine when solution programs are not over fit.
Test set	Final example data set used to measure final fitness of solution.

Table 2.1: Summary of GP Parameters

2.3 Object Detection in Computer Vision

Object detection is the task of finding "objects of interest" within images (*Localisation*) and determining the type, class, of the objects found (*Classification*). Typically these images will have been acquired using a digital camera, magnetic resonance imaging (MRI), satellite photography, or digital camera attached to a microscope, among others.

An image may contain many different structures within an image which the human viewer may perceive to be valid objects. For most images used within this report the distinction is clear between that of background image and the objects - all of which are considered "objects of interest". However, some images may contain many other objects which are unrelated to the task at hand. For the purposes of making the task simpler, these unrelated objects are considered to be part of the background image. Unless otherwise stated, whenever the word "objects" is used within this report, it is considered to mean "objects of interest" and to disregard any unrelated objects.

Object Localisation is the task of finding the positions of images to be classified. It is convenient to think of using the locations found as representing the positions to "cut-out".

These "cut-outs" are then "passed" to the classification stage. The cut-outs may be defined by the bounds of the area in which the classifier is to examine the image (centre and width and height), or they can be actual cut-outs taken from the original images and passed as smaller sub-images.

Object Classification requires having the positions of images and passing *cut-outs* of the objects to the classifying software.

2.3.1 Features

The features used for examining objects within images can very greatly and often can depend on the domain in which the images are acquired (eg: microscope imagery versus satellite imagery).

Technically, feature extraction is a process of reducing the number of potential dimensions needed to be handled by the learning system. A small cut-out of 10x10 has 100 pixels in total. If only a few data examples (of 10x10 pixels each) are needed, then this level of resolution can be tolerated. However, the number of feature values increases dramatically with larger cut-out sizes. A 100x100 pixel cut-out has 10,000 total pixels, which may be too large to handle for complicated problems requiring many training examples.

Extracting useful features reduces this dimensionality by attempting to remove unwanted or irrelevant data, to remove redundancy if it is present, and to simplify the learning task by approximating the data to some acceptable degree.

Past work has tended to focus on using hand crafted rules for feature extraction, or hand picking features which should be extracted and then training a system to detect these features [7, 12]. Some recent work has focused on using domain independent features [2, 5, 13, 11, 20, 18] such as statistics (mean and standard deviation) of regions.

2.3.2 Object Classification

The task of Object Classification is:

```
for each example of an object,
  classify its type as one of a set of possible classes.
```

The set of possible classes may or may not include a class representing "background". Including background as a class potentially makes the training harder, because the difficulty of learning is partially related to the number of classes, so background is only included when necessary.

Classification Strategies

The output from programs in GP are generally a single floating point value. Binary classification uses this value by considering values zero or greater as one class, and negative values as the other class. Using multiple classes (three or more classes) requires more complicated "classification strategies".

SRS: is the simplest of such strategies. All values below zero are considered the "background" class, and the positive values are statically broken into regions of range T, which is chosen by the user. This is illustrated by figure 2.6. Better methods have been developed more recently [14, 15] and the reader is encouraged to investigate these methods..

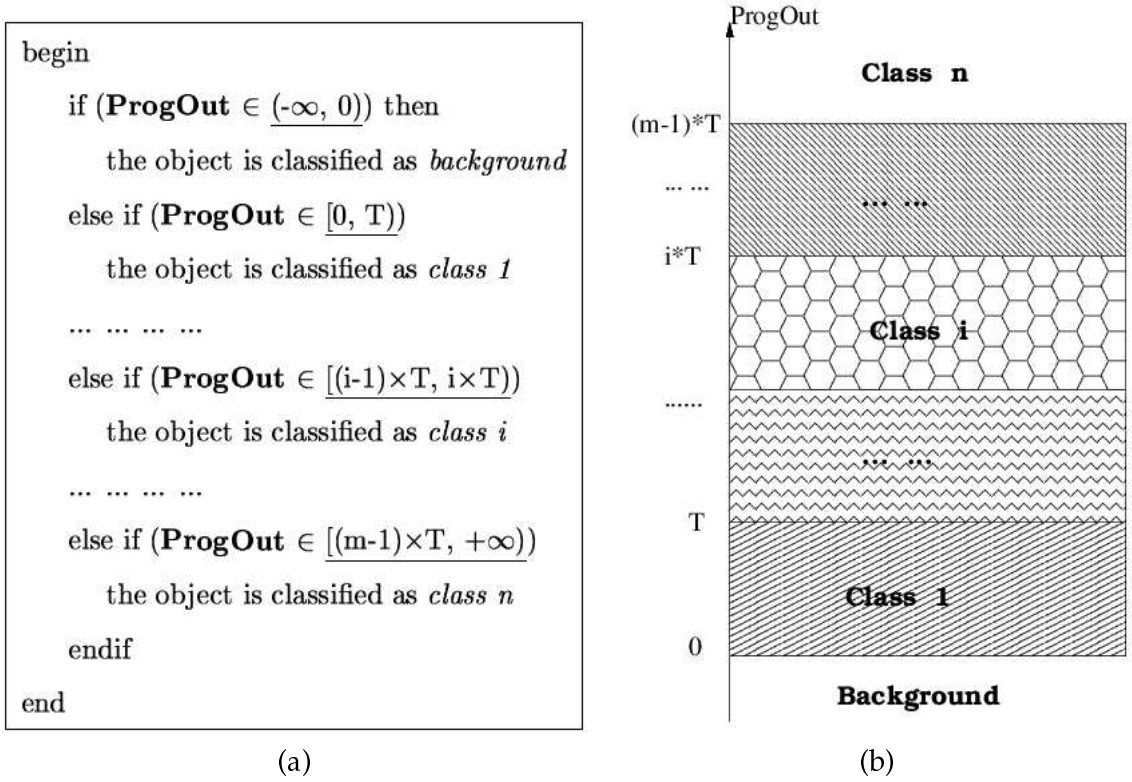


Figure 2.6: Static Range Selection[18, pp. 178]: (a) program section used to produce classification, (b) how value range is divided into classes.

2.3.3 Object Detection

Object Detection involves Object Localisation, followed Classification of these objects. Object Localisation is done as follows:

1. Pick relevant positions within an image, treating them as individual cut-outs (or examples), and
2. `for each cut-out,`
   `classify its type as one of:`
   `positive (an object of interest is here), or`
   `negative (no object here / background)`

In this way, Object Localisation is very similar to binary classification, but with the extra step of picking relevant positions to 'cut-out'. Consequently we can use similar measures of fitness. For example, precision and recall can be used to consider how many positions are "classified" correctly as either positive or negative.

Detecting Multiple Classes

Multiple object classes can be grouped into one "object of interest" class, and the localisation problem is treated as a binary classification problem over the set of cut-outs found using the sweeping window.

It should be noted here that this is still harder than a purely binary classification problem where the objects really are of the same class. Training the system to recognise multiple classes, which may be very different in nature, may require the system to learn disjunctive rules.

A different way to do this is discussed in section 2.4, however this is the approach taken in this report.

Sweeping Window

The relevant positions can be found using a sweeping window such as shown in figure 2.7. For training, the input window is moved across all positions of the training image (or images) and at each position the features are extracted. The correct position of all objects within the image are known and these positions are used to label the extracted features are representing an object or just background. The GP process produces a program learned from these labelled inputs.

During testing, a test image (or set of test images) is used in the same way. The input window is swept across the test image(s) and the features extracted. The program produced during training is then used to classify the features extracted from each input window position. The result is that all possible input window positions are classified as either "object of interest" (positive) or "background" (negative). The positively classified window positions are referred to as *Localisations*.

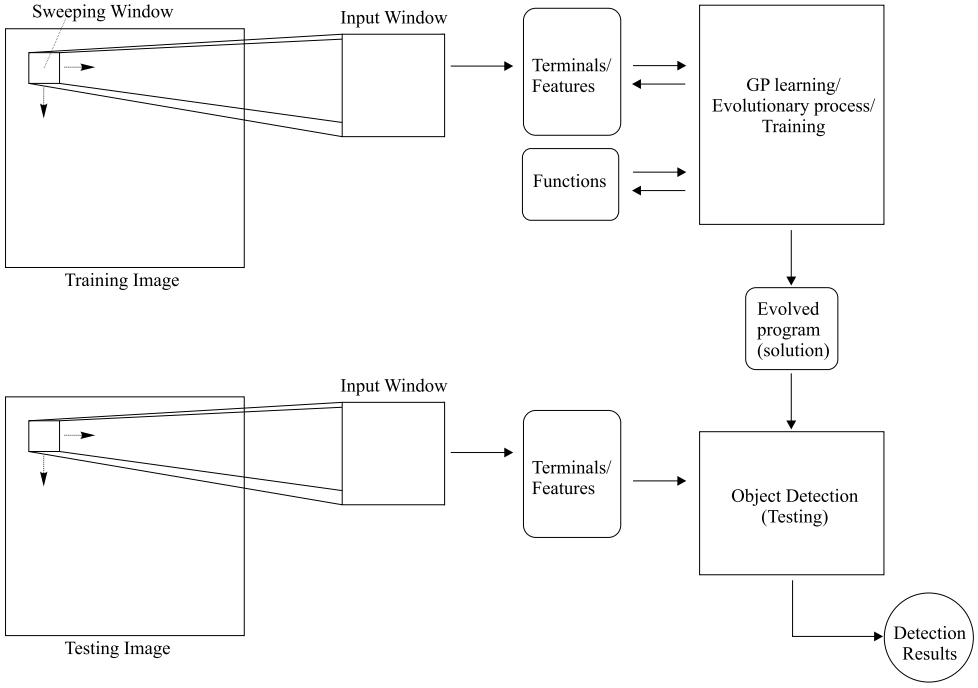


Figure 2.7: Sweeping Window

Clustering

The results from applying a localiser to the sweeping window can produce many individual localisation points for a single object. These individual positions tend to be grouped together in *clusters*. These clusters generally exist because the localiser was unable to pin-point the exact position of the objects. Before later stages, such as classification, can be applied to the localisation results, generally one needs to produce a single result for each cluster. These results thus produce the "centres" of clusters and can be used as the detected object position and supplied to an further processing stages.

The groups of localisation positions are collected using a *clustering algorithm*.

One clustering algorithm, devised for use in Neural Networks in [10] is also applicable to the detection problems using Genetic Programming. This method, called the *Donut Algorithm* groups adjacent pixels (individual localisation positions) repeatedly until no more pixels are added. The result from the Donut Algorithm is a list of points which represent the overall position of each separate cluster. Different positions can be used, such as the position half way between the minimum and maximum on both the x and y axis, or a more sensible "centre of mass" which is calculated by simply averaging all positions of individual localisations for a given cluster.

This clustering algorithm can process in time linearly proportional to the total number of pixels in the images. Any pixels which represent localisation positions and which are directly or diagonally adjacent to any other localisation positions become part of a larger cluster. It works by scanning each horizontal line in turn, moving progressively downwards. At each scan of a line, directly or diagonally adjacent pixels are grouped together and any information of prior associated clusters are merged into one. Finally, when all pixels for any clusters are left without adjacent pixels within the horizontal line immediately following them, those clusters which have not been continued are known to be searched in their entirety and they positions can be returned. During the processing, running averages and counters keep track of the number of pixels involved and the average position of these pixels. When a cluster has been scanned, the running averages are used to return the final averaged centre. Examples of this method working can be seen by examining the *detection maps* described in chapter 4.

The algorithm used here is very efficient when used prior to returning the final results, however it still takes time to compute the clustered results and incorporating clustering into a fitness function can slow the evolutionary process down considerably.

2.3.4 Measuring Performance

Under binary classification, some algorithm either returns *positive* or *negative*. The training data is then used to determine whether the classification was correct (*true*), or incorrect (*false*), giving four possible states: *true positive* (TP), *false positive* (FP), *true negative* (TN), and *false negative* (FN). Localisations of multiple classes are typed in the same way as for binary classification, where background is considered negative, and objects of interest as positive.

Accuracy

Accuracy is the percentage of classifications which are correct.

$$accuracy = \frac{number\ of\ correct\ classifications}{total\ number\ of\ classifications}, accuracy \in [0, 1] \tag{2.1}$$

This can also be represented using the true/false values:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

Standard Precision and Recall

Accuracy has a severe drawback when the ratio of negative to positive examples in the training set is biased. For example, if there 100 examples in the training set, only 5 of which are positive, then a classifier which classifies everything it sees as negative will have 95%

accuracy. For this reason, precision and recall are a much better measure. A classifier has *positive* precision and *positive* recall, calculated as follows:

$$\begin{aligned} precision &= \frac{number\ of\ correct\ positive\ classifications\ (true\ positives)}{total\ number\ of\ positive\ classifications\ made} \\ recall &= \frac{number\ of\ correct\ positive\ classifications}{total\ number\ of\ objects\ which\ should\ be\ classified\ as\ positive} \\ precision, recall &\in [0,1] \end{aligned} \tag{2.3}$$

'F' measure

The 'F' measure combines both positive precision and positive recall into a single value suitable for use in a fitness function:

$$F = \frac{2 \times precision \times recall}{precision + recall}, F \in [0,1] \tag{2.4}$$

Representing this using true/false fractions:

$$F = \frac{2TP}{2TP + FP + FN} \tag{2.5}$$

A full derivation is left to an appendix. It can be seen that this is very similar to accuracy, the difference being that the TN term is replaced with TP, giving TP the greater importance over TN. 100% F still is equivalent to 100% accuracy because FN combined with TP and FP make up for the lack of TN and together mean that 100% F does not allow a classifier to miss having the correct number of TN.

Abstract and Concrete Concepts of Object Localisations

At an abstract level, one can talk about "the precision and recall percentages of object localisations" as referring to the ability to find the existence of objects at some set of approximate locations, but without any specific coordinates given. The intuitive concept is to consider whether the localisation resided within the boundaries of the object. A localiser thus has high precision if its localisations mostly reside within the bounds of the objects it has found.

However, objects have have width, height, radius and perhaps arbitrary shape, and most systems require a more exact position to be provided for the object of interest. A typical measure is whether the localisations are within some tolerance of the exact, or *true*, centres of the objects. A set of localisation positions will thus only have high precision if most are with the tolerance of true centres, and these localisation positions will have very low precision if most reside within the objects' bounds but not at the exact centres. In this scenario, a localisation within the object bounds is considered a "false positive" if it is not at the exact centre. In other words, it is treated as if it is just as bad as picking some point on the background and calling it an object of interest.

In chapter 4 this report introduces another version of precision, recall, etc., which closely resembles the intuitive sense of object bounds, but at the same time prefers programs to find object close to their centres.

The remainder of performance measure descriptions are for the task of object detection/localisation specifically.

Average Position Error

Average positional error is the average distance of detections/individual localisation positions from object centres, and considers only the detections/individual localisations made within object bounds, ie: within the radius of the object.

2.3.5 Detection Rate

Detection Rate (DR) is simply the percentage of objects detected, or the number of objects successfully detected within the desired tolerance from true centre, divided by the total number of target objects. It is equal to the Recall, but is defined here as it would be for multiple class object detection:

$$DR = \frac{\sum_{i=1}^{n} N_{true}(i)}{\sum_{i=1}^{n} N_{known}(i)} \times 100\% \tag{2.6}$$

where:

- n 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,

2.3.6 False Alarm Rate

False Alarm Rate (FAR) is the total number of false alarms, taken as a percentage of the total number of target objects. It is generally expressed as either a percentage, or as a fraction of number of false alarms per object. The false alarm rate is defined as:

$$FAR = \frac{\sum_{i=1}^{n} N_{reported}(i) - \sum_{i=1}^{n} N_{true}(i)}{\sum_{i=1}^{n} N_{known}(i)} \times 100\% \tag{2.7}$$

where:

- $N_{reported}$ is the total number of objects reported for class i, whether correct or incorrect.

False Alarm Area

False Alarm Area (FAA), introduced by Pritchard and Zhang in [11], is defined as the number of false alarm pixels or number of individual localisation positions which are not at the centre of correct clusters, divided by the number of objects. It is approximately equal to the average area of all clusters. It is reported as the number of false alarm pixels per object.

2.4 GP for Object Detection

The object detection and classification systems to date has typically involved only two classes: the objects of interest, and the background. Some work [2, 13, 11, 18] has been done on classification of multiple classes, and even less on [2, 11, 18] has been done on detection of multiple classes.

This project focuses on the problem of locating objects of multiple classes, and of efficiently producing programs which can perform this task with high detection rate and low false alarm rate and positional error.

2.4.1 Object Detection Method

Formally, object detection involves both localisation and classification. In the past, detection has been performed using multiple stages of image processing. Each successive stage was usually domain dependent and hand crafted to fit the problem domain. Such stages could include: preprocessing, segmentation, filtering, feature extraction, and finally object detection as illustrated it figure 2.8. [18] defines *multiple stage* detection methods as using each succesive stage to refine or to continuously improve the results from one stage to the next.

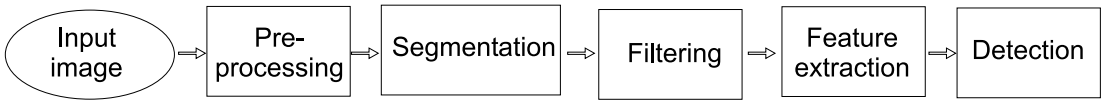


Figure 2.8: Typical multiple stage detection method

More recent work [18, 11, 5, 2, 10] has rejected this approach on the basis that its success relies too heavily on the results of each successive stage. If one stage looses information, that information will be lost permanently and the final stage will only yield the detections of a fraction of the total original targets.

As such, rente work, particularly in GP and Neural Networks, have focussed on combining all stages into one. This one stage is learned entirely through some learning process such as GP or Neural Networks. In particular, both localisation and classification are performed at once: the output being a class label for each position of a sweeping widnow, as illustrated in figure 2.9. This attempts to elliminate any problems of loosing information by performing all tasks in the single stage.

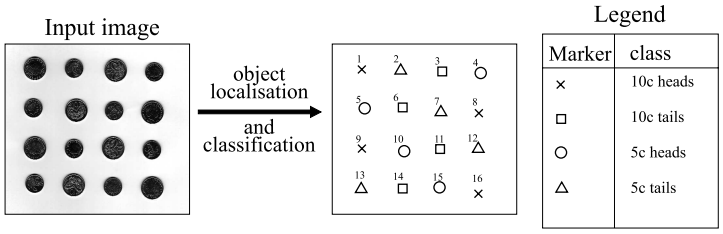


Figure 2.9: Single stage detection method

However, if using GP a fitness function can be hard to devise to achieve good results as it needs to consider both the classification accuracy and how close the found positions of objects are to the objects' true positions (positional accuracy). The latter of these two problems is typically done by the use a fixed TOLERANCE value [11, 10, 2]. The tolerance defines a square region around the true centres of objects: $(x \pm \text{TOLERANCE}, y \pm \text{TOLERANCE})$. If the position of an object is detected to be within that region of the object's true centre, (x, y), then the position is considered to be correct.

The former issue, of classification accuracy, is dealt with by rejecting any positionally correct detections which were classified incorrectly. The final fitness is thus taken from the number of correct classifications at positions within TOLERANCE pixels of an object of that classified class, the number of incorrect classifications, from the detected positions outside of the TOLERANCE range, and also from the number of objects missed. Specifically, the fitness functions include the Detection Rate (DR), the False Alarm Rate (FAR) and the False Alarm Area (FAA).

With the programs typically produced by GP, the detection problem reduces to a classification problem, only that there are a great many more cases of background than objects. The problem is reduced to classification because the genetically evolved programs are used by being applied to the features extracted from the sweeping window and in effect it is classifying the features extracted at each point that the window moves to.

2.4.2 Fitness Functions

Some past research has focused on the use of separately calculated measures which are separately combined within a weighted sum:

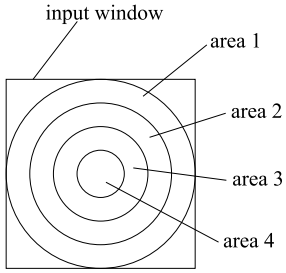
- Detection Rate (DR), the percentage of objects detected.
- False Alarm Rate (FAR), the percentage of incorrect localisations.
- False Alarm Area (FAA), used by [11], measures the number of multiple extra localisations of objects.
- Program Size, used by [2], measures the depth of programs in attempt to produce smaller and less redundant programs.

This project focuses on the use of the 'F' measure because it has some nice features in the way that it combines precision and recall:

- Precision and recall are given equal importance.
- Preference is made to having similar values for precision and recall, over having opposite extremes for precision and recall (ie: one high, one low).

2.4.3 Features

One suggestion for the features is to use a series of concentric circles[2], such as illustrated in figure 2.10. The features are extracted by calculating the mean and standard deviation of pixel values within each of the circular areas. This set of features has the advantage of being *rotationally independent*, meaning that the rotation of the object does not affect the feature values. It is, however, *scalar dependent*, meaning that the scale of the object affects the feature values.



Mean	SD	Regions
F1	F5	Circular Area 1
F2	F6	Circular Area 2
F3	F7	Circular Area 3
F4	F8	Circular Area 4

Figure 2.10: 8 concentric circle features[2, pp. 25]

2.5 Related Work

The following serves as a brief literature survey of some of the areas relevent to the topics in this report. Firstly the various methods relating to fitness functions are features are presented, followed by some other work related to object detection. Finally some examples of learning/training theories relevent to our work is presented.

Various research has been done on image processing, Genetic Programming, and GP applied to object detection specifically. Some research has been done using domain independent approaches of classification, using pixel values and statistics of pixels values. Some of these[2] have divided a fixed sized input window into circular or rectangular regions over which statistics are calculated.

2.5.1 Fitness Functions

Zhang et al.[21] developed and compared two feature sets for object detection based on domain independent pixel statistics and also the use of naw pixels. The first feature set divided a square input field into four regions and a fifth overlapping region in the centre. It used the mean and standard deviation of all pixels within the five regions and along four boundary lines. The second used the mean and standard deviation of concentric circle boundaries. Here, if the input field is $n \times n$ in size, then $\lceil \frac{n}{2} \rceil$ concentric circles are used, with radiuses incrementing by a single pixel between each.

The fitness function used detection rate and false alarm rate and combined them in the following equation:

$$\text{fitness}(FAR, DR) = W_f \times FAR + W_d \times (1 - DR)$$

Pritchard and Zhang[11] investigated the use of GP for object detection of multiple classes using a single stage detection method and similar data sets to the ones use in this project. He introduced a new term in the traditional fitness function for object detection: False Alarm Area (FAA). Clustering of localisation positions was used. It was found that Detection Rate and False Alarm Rate were insufficient to measure small changes in fitness whereby the size, or area, of clusters was smaller from some program than another. FAR only counts the number of false alarm clusters. So FAA measured the total number of pixels which were involved in false alarm clusters and the extra number of individual localisation positions, other than true centres, for the true positive clusters.

For example, figure 2.11(b) is better than figure 2.11(a) because there are fewer false alarm pixels. The new fitness function was found to produce very good results. 100% classi-

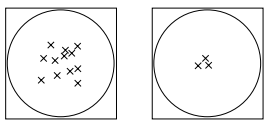


Figure 2.11: FAA examples: (a) many false alarm pixels, (b) few false alarm pixels

fication accuracy was achieved in most cases and around 100% of objects where successfully localised to within a tolerance of two pixels plus or minus from the true object centres. The new fitness function contained three free parameters which need experimentation to set. Typical values used were not specified. Pritchard's fitness function was:

$$fitness(DR, FAR, FAA) = a \times (1 - DR) + b \times FAA + c \times FAR$$

Bhowan and Zhang[2] compared the use of two fitness functions based on the same principles as used in papers already mentioned. The first was the traditional Detection Rate and False Alarm Rate based fitness function. The second extended the work of [11] to include a fourth term: program size. The intention of the program size term was to reduce the overall program size and hopefully produce simpler programs which are easier to interpret if later examined. Bhowan used a single stage object detection method.

The two fitness functions were:

$$\text{fitness}(DR, FAR) = K1 \times (1 - DR) + K2 \times FAR$$

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

Bhowan investigated three methods of training for detection and classification, the last of which incorporated two training phases. The first method, Straight Forward Detection (SFD), trained programs by directly applying them to the detection problem in the typical fashion as described in detail in 2.3.3.

The second training method, Object Classification Applied to Detection (OCAD), trained using only the object classification training data which contains mostly examples of training cut-outs centred on objects and of background. The training data also includes some specially selected examples which overlap background and objects. This method is primarily used because it uses substantially less training examples than SFD.

The third method, Refining Object Classification for Detection (ROCD), is a combination of the first two methods, and uses two training phases. The first phase trains as for the training of the OCAD method for some number of generations, generally until perfect classification accuracy is achieved. The second phase then continues the training of the programs produced by the first phase by applying them to object detection training data like in SFD. The training is continued until a reasonable detection rate and false alarm are achieved.

The second of the two fitness functions was found to produce equally successful programs to the first fitness function, with less training generations and the program sizes were around a third that of the programs produced using the first fitness function.

The ROCD method was found to have the best training performance for good detection effectiveness, however SFD produced fewer false alarms.

Both fitness functions used here involved free parameters which need to be set by experimentation. The typical values used for the coins data set in their report are as follows:

K1	=	5000
K2	=	100
K3	=	10
K4	=	1

2.5.2 Other Object Detection Work

Ny and Zhang[10] investigated the use of Neural Networks for object detection. A clustering algorithm called the "Donut Algorithm" was used to cluster groups of potential localisation positions into a single central point and then this result was used to filter false alarms to determine whether the cluster represented a true object of a given class or a different class.

The clustering method used in this report, and described later, is as described in their work on the "Donut Algorithm" in [10].

Schneiderman and Kanade[12] used statistics of image parts for multi-class detection of objects. Multiple, individual classifiers where used to manage the separate object classes, and also for a discrete set of sizes. Each classifier was based on the statistics of localised parts. These parts were transformations taken from subsets of wavelet coefficients, and

yielded discrete sets of values for each part. Each classifier determined its result using a process similar to a Bayesian Network. They developed a process which could detect human faces, cars, and door knobs from many different angles and scales and classify the human faces as facing left, right or towards the camera.

2.5.3 Training Set Theories

Boonyanunta and Zeephongsekul[4] investigated the effect of training set size on the predictive power of learned classifiers. The predictive power is the ability of a learning system to generalise and to produce good results when applied to some test data set. Three stages of predictive power growth were observed. In the first stage, as the number of training examples is increased, the predictive power increases rapidly. In the second, the power continues to increase, but at a much slower rate. In the third, the predictive power more or less plateaus for increased training size. They proposed an equation which models the effectiveness and can be used to predict an estimate of the the training size required to achieve a desired level of accuracy:

$$P(p) = T(1 - \exp^{-kp}) + P(0)\exp^{-kp} \tag{2.8}$$

where:

- $P(0)$ is the predictive power without training data (for example, a random coin toss has 0.5)
- k is efficiency rate, namely: the rate of improvement in predictive power per unit increase in efficiency.
- p is the training sample size.

There is a theory [17] for Neural Networks that the number of training examples required to train a network to a given accuracy factor, ϵ, with N weights within the network, is approximately equal to $N\frac{1}{\epsilon}$. Where ϵ is the maximum allowed error rate. For example, if we require 90% accuracy (an ϵ value of 0.1) we will need 10 times as many training examples as weights to achieve the desired level of generalisation.

[1] found that learning algorithms could be augmented to attempt to minimise the number of non-zero weights and thus discard some unnecessary nodes. From this they theorised a lower bound on the number of training examples required. This lower bound is proportional to the number of hidden nodes, instead of the total number of weights. Their theories were applied to boolean threshold transfer functions.

[16][ch. 11] includes a section highlighting the existing theories and "rules-of-thumb" for selecting a good training set for a Neural Network learning system.

1. The training set should be statistically representative of the problem domain: the probabilities of different types of training examples should be approximately equal to the likelihoods of finding these types of examples in the application. As an example, [16][pp. 224] describes a situation where some learning method is applied to men of different heights. If the training set contains the majority of examples from men of short stature, then the trained classifier will produce inaccurate results for taller men. And vice-versa.

2. Where possible, training sets should be from objective data, rather than subjective data. Subjective data is based on the opinions or guesses of some expert, but may be contradicted or disagreed upon by some other accurate expert. Objective data is generally more accurate and is obtained by some well defined rule. However, it is pointed out, objective data may be impossible to acquire for some problems.

[16] also highlighted theories regarding normalisation of training data, generalisation theory relating to the number of hidden nodes to training set size as discussed by [17], and the VC dimension as described by Blumer et al. in [3].

2.5.4 Summary of Fitness Functions

Three typical fitness functions have been found in the previous related work search:
[21, 19, 2] fitness(DR,FAR) =

$$K1 \times (1 - DR) + K2 \times FAR$$

[11] fitness(DR,FAR,FAA) =

$$K1 \times (1 - DR) + K2 \times FAR + K3 \times FAA$$

[2] fitness(DR,FAR,FAA,ProgSize) =

$$K1 \times (1 - DR) + K2 \times FAR + K3 \times FAA + K4 \times ProgSize$$

And some suggested values for K1 to K4 are:

K1	=	5000
K2	=	100
K3	=	10
K4	=	1

Chapter 3

Tasks

3.1 Data Set

The methods developed for the goals described in section 1.2 have been tested using four data sets. The first is relatively easy, the second and third are equally harder, and the last is much harder by combining the second and third data sets. All data sets contain gray scale images. In all data sets, the largest object has a diameter of 70 pixels, and so the "sweeping window" size is set to 70x70 pixels.

Easy coins. The easiest training set. It contains images of 5 and 10 cent coins on a fairly uniform light background. There are 24 images in this set, each sized around 540x540 pixels. There are five classes in this set: background, 10c heads, 10c tails, 5c heads, 5c tails. See figure 3.1(a).

5cent coins. One of two medium difficulty training sets. It contains images of 5 cent coins only, on a noisy dark background. There are 24 images in this set, each sized around 540x540 pixels. There are three classes in this set: background, 5c heads, 5c tails. See figure 3.1(b).

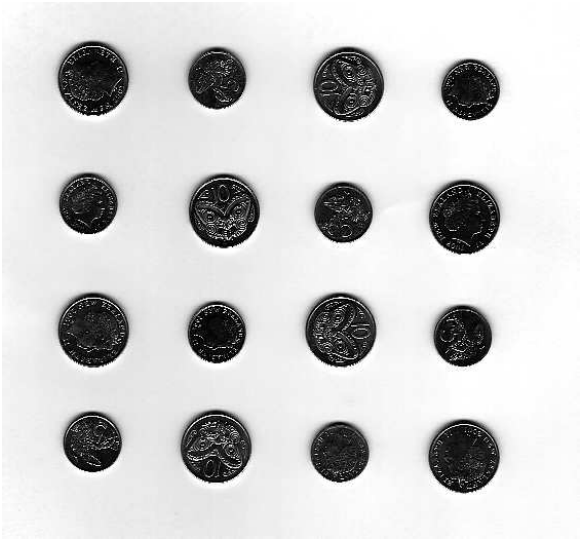
10cent coins. Second of two medium difficulty training sets. It contains images of 5 cent coins only, on a noisy dark background. There are 24 images in this set, each sized around 540x540 pixels. There are three classes in this set: background, 5c heads, 5c tails. See figure 3.1(c).

Hard coins. The most difficult training set. It contains images of 5 and 10 cent coins on a noisy dark background. For this set, half of the images from the '5cent coins' set are alternated with half from the '10cent coins' set. This gives a total of 24 images, each sized around 540x540 pixels. There are five classes in this set: background, 10c heads, 10c tails, 5c heads, 5c tails.

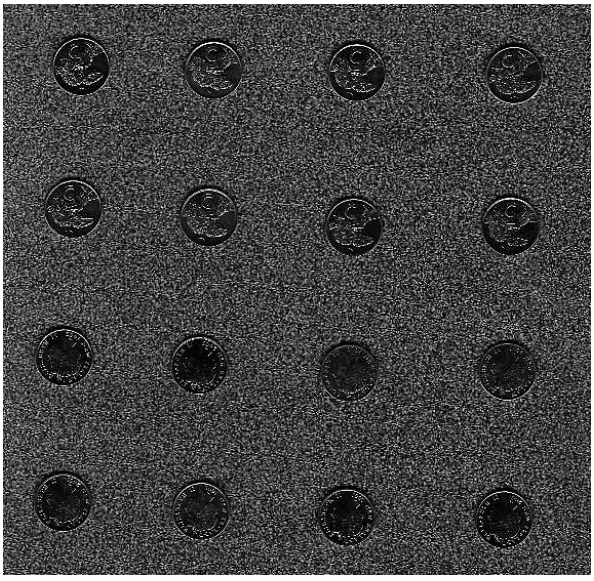
3.2 Tasks

In each data set, the tasks are to:

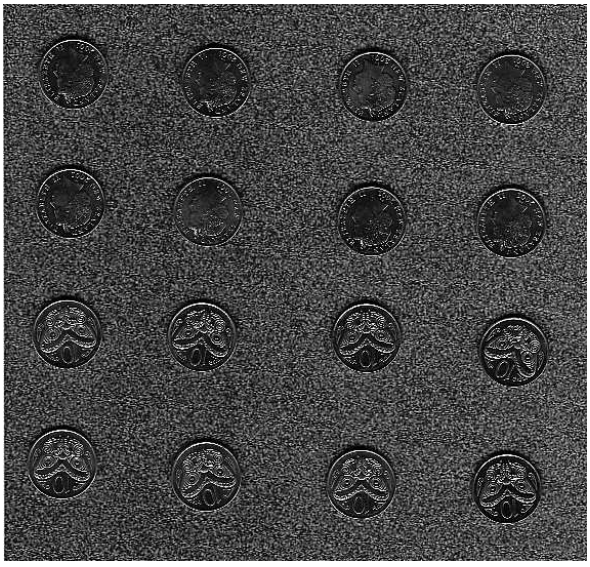
1. maximum precision and recall,
2. achieve high positional accuracy, and
3. minimise the number of extra localisations per object.



(a)



(b)



(c)

Figure 3.1: Data Sets: (a) Easy coins, (b) 5cent coins, (c) 10cent coins.

Chapter 4

New Fitness Functions

In this chapter we seek to answer the first two research goals by devising a fitness function for object localisation that can be easily used without the need for finding good values for many free parameters, and which is efficient to calculate without the need for computationally expensive clustering of results. We compare our results against an existing method and show that our new approach to be extremely competitive, producing similar detection rate with fewer false alarms and less training time.

For simplicity, we consider objects of interest which are approximately round. We later suggest that our new method may be sufficient to handle most detection problems.

4.1 Object Detection Method

Chapter 2 introduced two Object Detection methods used in past research. A different approach has been taken in this project. Similar to the multiple stage method discussed, the final result is found after two separate stages. The difference is that both stages are performed by evolved genetic programs or some other learning method, such as Neural Networks, which can be trained in a fashion independent of the object detection problem.

The first of the two stages used here finds only the locations of the target objects. The goal for this stage is to achieve high positional accuracy and high recall. The result of this stage has clustering applied to find the centres of "patches" of localisation positions and these centres are passed as inputs to the classification stage. The second stage performs classification. The positions found in the first stage are used to extract what can be thought of as "cut-outs" at each of these positions. These cut-outs are classified using an evolved/trained classifier independent of the evolved/trained localiser. Figure 4.1 illustrates this approach. The first box represents the raw image, the second shows the resultant positions of localisation and clustering, the group of cut-outs are extracted and then each is individually classified and labelled with its class. The positions will also be associated with each object.

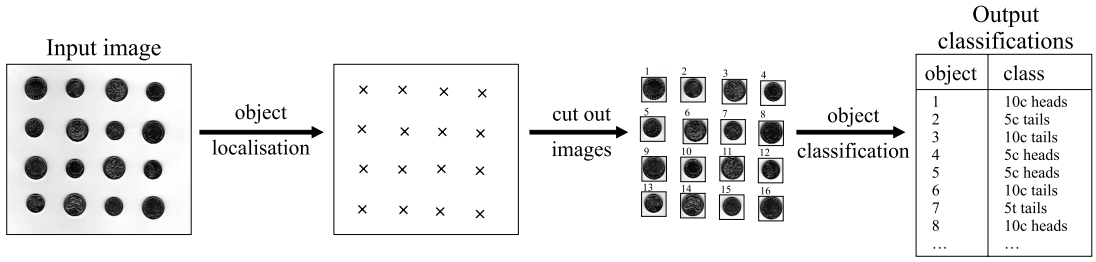


Figure 4.1: Two stage detection method

The following serves as a comparison of some advantages and disadvantages of the single and two stage detection methods.

Single stage

- Only one program to train.
- Hard to train because goal is very specific and restricted: correct position and correct class.
- Tends to achieve high recall.
- Potentially could tend to produce bad classification accuracy.
- Generally able to accept greater positional error, because the classification is performed also.

Two Stage

- More than one set of training: takes longer, have to devise two separate fitness functions.
- Easy to train each stage, fitness functions are simpler and easier to devise,
- especially classification, which is easy to train and achieve high classification accuracy.
- Generally the localisation stage must produce high positional accuracy so that the classification stage can be successful.
- Tends to achieve low recall because of requirement for low positional error.
- High classification accuracy possible, see [14].

It is unlikely that a single stage localisation and classification detector will produce better recall than one trained for localisation only. The localisation only method is required to do less and so should be better at it. Consequently, there should be no more information loss during the localisation stage than would be found in an equivalent single stage program. The only potential information loss comes from a more subtle difference.

A single stage approach will produce multiple individual localisation positions for a single object, each with an assigned class. These classes may not agree and a "majority rules" approach should be applied (although the literature observed so far does not seem to allude to this point). Here, if for example one object is localised four times as in figure 4.2, with one being assigned class 1, and the other three being assigned class 2, the clustering algorithm should group these as one and assign it class 2.

The two stage method performs clustering before classification is performed, and thus the class is calculated from features extracted at a single position, unlike the single stage method which classifies all pixels. It is thus possible that the localisation stage could determine the object centre to be at the position of the bottom right pixel in figure 4.2, and consequently the classifier would assign the wrong class label.

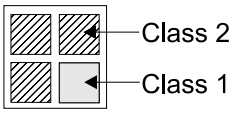


Figure 4.2: Example classification of four adjacent pixels

4.2 Issues for Localisation Fitness Functions

Here the requirements on fitness functions for classification only are compared to the requirements for localisation. So that we can get a clear picture of the what we are asking for. A fitness function for classification needs only to consider four possibilities:

1. True Positive, where the classifier correctly classified an object as positive (object of interest),

2. False Positive (also called False Alarms), where the classifier incorrectly classified the background as positive,

3. True Negative, where the classifier correctly classified the background as negative, and

4. False Negative, where the classifier missed an object, instead classifying it as negative (background).

Typically, a simple calculation of the percentage correct is sufficient for a measure of fitness for classification. Or, if more control is desired, precision and recall can be used.

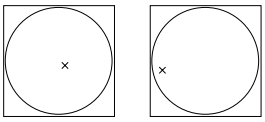
Object localisation is similar to object classification because it requires classifying the image surrounding a single point within the image and doing this for every point as sampled by the sweeping window. A fitness function for object localisation must thus handle the four cases mentioned for classification, however it must also handle another situation:

> "Assuming we define an object's position as its exact centre, should a point within an object, but not at its exact centre, be classified as object or background?"

In practice we only want the exact centre, but it can be dangerous to accept only these and reject all the rest as false positives. For one reason, this disregards how close a localisation is to the exact centre. Thus, if programs localise objects to a position very close to their exact centres, these localisation positions will be treated as false alarms and consequently these programs will be considered to be equally unfit to other programs which tend to produce false alarms on the background. There are two problems with this:

1. As already mentioned, programs which tend to localise objects close to their object centres are considered no better than programs which tend to localise objects further from the object centres. For example, the two example detections illustrated in figure 4.3 may both be considered false alarms as neither are at the object's centre.

2. With the requirement that programs find exact centres comes the requirement that the training data is perfect. This is very difficult for real world situations where object centres may not be entirely clear, and often humans must pick the object centres manually, often with mistakes. Put simply, the training data is not perfect, so the resultant programs are going to be confused. For example, it would be very bad if the training data included many examples like that in figure 4.3(b).



(a) (b)

Figure 4.3: Example positions of detections of a single round object (the small crosses represent the detected position)

The localisation stage should have high recall whenever possible, and it may be possible to train classifiers to classify cutouts which are not perfectly centred on objects but have a small positional error. Thus, a fitness function which accepts small positional errors will produce better localisers than others for this task. Generally the solution to this has been the use of a small tolerance. If objects are localised to within, say, three pixels of their true centres, then the object localisations are considered correct.

4.3 Localisation Fitness

We wish to incorporate some measure of positional accuracy into our fitness functions which can give individual localisations some value of correctness which ranges from bad to perfect. This developed into a weight, which we called the "Localisation Fitness". This is separate to the full fitness function of the evolved genetic programs associated with training localisers, instead this measures the relative fitness of an individual localisation position.

Under our new fitness functions, each localisation position of an object is considered to be "correct" (or "true") if it was localised within some radius of the object's true centre. The radius used may be the radius of the object itself, or it may be a fixed radius picked from a priori knowledge. This radius is discussed further later and it is referred to as the "Localisation Fitness Radius", or just "Fitness Radius" for short. The radius used in the experiments discussed later was the radius of the largest object of interest (35 pixels), and also equal to half the width of the square sweeping window.

Each correct localisation is given a "localisation fitness". This fitness is some value between 0.0 and 1.0, where 1.0 is best, and 0.0 is worst. If all correct localisation fitnesses are 1.0, then the program is picking only exact centres as positives and thus its "precision" is 100%. If all correct localisation fitnesses are 0.0, then the program is picking all objects well off their true centres and thus it is equivalent to it having a precision of 0%.

The formula used to produce this fitness can be any reasonable function, the choice thereof affecting slightly the characteristics of the overall fitness function. The function chosen here is a linear one such that localisations made within the radius of the object are given a value proportional to the distance from the centre (1.0 at exact centre, 0.0 at the very edge). Any localisations made outside this range are given 0.0. Specifically:

$$\mathit{localisationFitness} = \begin{cases} 1 - d/r & \text{if } d \leq r \\ 0.0 & \text{otherwise} \end{cases} \tag{4.1}$$

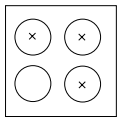
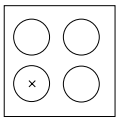
where:

- d is the distance from object centre
- r is the localisation fitness radius (half width of sweeping window in tests done to date)
- *localisationFitness* is the weight given to this particular correct localisation when calculating the precision and recall as discussed below.

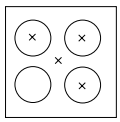
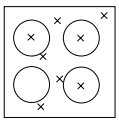
4.4 Fitness Functions Design Considerations for Localisation

We examined the requirements on fitness functions for localisation problems and found that the fitness function should order different programs in the following way:

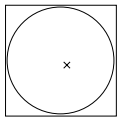
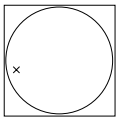
1. Greater number of objects should be localised (their positions found).

 BETTER THAN: 

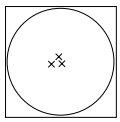
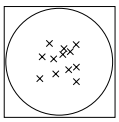
2. Fewer numbers of false alarms on the background should be preferred.

 BETTER THAN: 

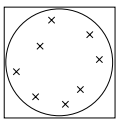
3. Localisations should tend to be closer to exact centre than further away.

 BETTER THAN: 

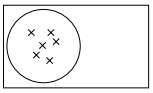
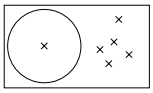
4. Fewer separate localisations of a single object should be preferred.

 BETTER THAN: 

5. If an object is given multiple localisations, then these should tend to be closer to the object's exact centre, they should not be spread out (this makes the results after final clustering better).

 BETTER THAN: 

6. There is no obvious preference of multiple additional localisations of objects over background false alarms unless you consider the effect of the final clustering stage, which should collect all additional localisations into one. Consequently it may be best that the fitness function prefers multiple localisations over background false alarms.

 EQUAL/BETTER THAN?: 

4.5 The First New Fitness Function – LFWF

We now describe the equations for the first of two new fitness functions. *Localisation Fitness Weighted 'F' Measure* (LFWF) incorporates precision and recall for their ability to easily handle the first two of the fitness function design requirements noted previously, and we introduce weights which handle requirements 3 to 5.

Typical precision and recall are only used for discrete values: correct or incorrect. In order to use the localisation fitness weight the formulas for precision and recall are modified. We consider that counting the number of true positives is equivalent to summing the number 1 for each true positive. For example, for n true positives:

$$\textit{number of true positives} = \sum_{i=1}^{n} 1$$

We can replace the '1' with a weight, such as in the following:

$$\textit{number of true positives} = \sum_{i=1}^{n} weight_i$$

We also need to handle the separate localisations of each individual object. It turns out that a good method is to take the average localisation fitness of localisations for each object. A full explanation of the reasoning behind this can be found in appendix A. The precision and recall are combined using the 'F' measure. The final equations involved are as follows:

$$\text{weightedPrecision} = \frac{\sum_{i=1}^{N} \text{ave}(\{\text{localisationFitness}_{i,j}, j = 1..L_i\})}{L} \tag{4.2}$$

$$\text{weightedRecall} = \frac{\sum_{i=1}^{N} \text{ave}(\{\text{localisationFitness}_{i,j}, j = 1..L_i\})}{N} \tag{4.3}$$

$$\text{fitness}_{LFWF} = \frac{2 \times \text{weightedPrecision} \times \text{weightedRecall}}{\text{weightedPrecision} + \text{weightedRecall}} \tag{4.4}$$

where:

- N is the total number of target objects in the data set,
- L is total number of localisations made by a genetic program,
- $\textit{localisationFitness}_{i,j}$ is the localisation fitness of the j-th localisation of object i,
- L_i is number of localisations made to object i,
- *weightedPrecision* is the precision in range $[0, 1]$, 1.0 means all localisations where correct, with no extra localisations of objects,
- *weightedRecall* is the recall in range $[0, 1]$, 1.0 means all objects are localised, and
- $\textit{fitness}_{LFWF}$ is in range $[0, 1]$, 1.0 means perfect fitness: all localisations are at the exact object centres with no extra localisations of objects and no false alarms.

4.6 The Second New Fitness Function - APWF

Average Position Weighted 'F' Measure (APWF) is based on the same method as LFWF, with an exception in how the localisation fitness for each object is calculated. In LFWF the localisation fitnesses of all localisations for an object are averaged and that value is used in the sum for the weighted precision and recall. In APWF the positions of the localisations for an object are averaged, and the individual localisation fitness is calculated for this point only and this is used as the value in the sum instead.

The reasoning behind this relates to the way that clustering methods decide the final position. The clustering method used in this project takes the average of all points found within a cluster, and uses this as the final position of the cluster. Incorporating the average position within our fitness function mimics this characteristic.

The equations for APWF are as follows:

$$\text{weightedPrecision} = \frac{\sum_{i=1}^{N} \text{localisationFitness}(\bar{x}_i)}{L} \tag{4.5}$$

$$\text{weightedRecall} = \frac{\sum_{i=1}^{N} \text{localisationFitness}(\bar{x}_i)}{N} \tag{4.6}$$

$$\text{fitness}_{APWF} = \frac{2 \times \text{weightedPrecision} \times \text{weightedRecall}}{\text{weightedPrecision} + \text{weightedRecall}} \tag{4.7}$$

where:

- N is the total number of target objects in the data set,
- L is total number of localisations made by a genetic program,
- *localisationFitness*$(\bar{x}_i)$ is the localisation fitness of the average position over all localisations for the i-th object. In other words, take all the localisation positions for the i-th object, average their x and y coordinates, and use this coordinate to calculate the localisation fitness for the i-th object.
- *weightedPrecision* is the precision in range $[0, 1]$, 1.0 means all localisations where correct, with no extra localisations of objects,
- *weightedRecall* is the recall in range $[0, 1]$, 1.0 means all objects are localised, and
- *fitness*$_{APWF}$ is in range $[0, 1]$, 1.0 means perfect fitness: all localisations are at the exact object centres with no extra localisations of objects and no false alarms.

4.7 An Existing Fitness Function

We compared the performance and effectiveness of the two new fitness functions with an existing method which uses clustering before calculating the fitness and was used by Pritchard et. al. in [11]. Pritchard found that best results where achieved using Detection Rate (DR) and False Alarm Rate (FAR) along with False Alarm Area (FAA). We refer to their fitness function as *Clustering Based Fitness*(CBF).

The equation for the fitness function calculations is as follows:

$$fitness(DR, FAR, FAA) = K1 \times (1 - DR) + K2 \times FAR + K3 \times FAA$$

number of gens	50
initial population	500
mutation rate	30
elitism rate	10
maximum program depth	6
minimum program depth	3
function set	$\{+, -, \times, \%, \text{if}<0\}$
fitness functions	LFWF, APWF, CBF
training set size	3968 (samples taken from first 8 images in data set)
validation set size	3968 (samples taken from second 8 images)
test set	full sweep of last 8 images
K1	5000
K2	100
K3	10

Table 4.1: GP parameters used

4.8 Experimental Setup

To compare the fitness functions we apply them to object localisation training on all four data sets and measure their training performance and test set detection rates and false alarm rates. A maximum of only 50 generations are evolved, during which the best program from each generation is applied to the validation set to avoid over fitting. After all 50 generations have completed, the program which scored the best fitness on the validation set is taken as the final program. It is then applied to the full test data set and its performance and effectiveness measured. By restricting the evolutionary process to only 50 generations we will likely give the programs sufficient time to evolve workable results but which are not as good as could be achieved if given a much longer training time. This allows us to measure well the ability of the fitness functions to quickly produce good genetic programs.

The basic genetic parameters, other specific parameters such as the K1, K2, K3 values for CBF, and details of training data have been set as follows listed in table 4.1.

4.9 Results And Analysis

This section presents results comparing the two new fitness functions to the existing fitness function. Table 4.2 presents the main results after final clustering has been applied to the individual localisation positions and the cluster centres compared to the target object centres. If an object is detected to within a TOLERANCE of three pixels of the true object centre, that detection is considered correct and improves the detection rate, otherwise the detection is considered a false alarm. Each row within the table represents results averaged over 100 experiments. The training performance columns represent the number of generations and the time taken to train to that generation when the best validation set results were achieved. The test accuracy columns measure the detection rate (DR), false alarm rate (FAR) and false alarm area (FAA) when those best validation set programs where applied to the test set.

For example, the first row informs us that when LFWF is used to evolve programs for the easy coins, best validation set fitness was achieved, on average, in 38.11 generations, taking 113.38 seconds of training time. When applied to the test set, the programs detected 81.04% of all target objects (within 3 pixels of true centre) with 48.52% false alarm rate (0.49 false

Dataset	Fitness function	Training Performance		Test Accuracy		
		Generations	Time (s)	DR (%)	FAR (%)	FAA
Easy Coins	LFWF	38.11	113.38	81.04	48.52	98.35
	APWF	36.44	111.33	82.05	43.42	123.17
	CBF	13.69	178.99	81.88	92.25	324.09
5 cent Coins	LFWF	37.52	118.45	54.50	135.81	103.79
	APWF	38.27	121.15	54.39	157.30	145.83
	CBF	31.74	432.05	51.66	229.49	930.19
10 cent Coins	LFWF	34.35	107.18	52.88	63.18	95.69
	APWF	37.88	123.77	50.41	136.16	163.65
	CBF	36.90	493.94	31.08	192.04	804.88
Hard Coins	LFWF	33.27	105.56	27.34	92.77	114.86
	APWF	37.27	118.13	29.34	185.51	187.50
	CBF	31.02	431.65	27.54	220.73	1484.51

Table 4.2: Detection Accuracy of all fitness functions on coin datasets

alarms per object, averaged over all objects), and 98.35 false alarm pixels per object (area of clusters averaged over the number of objects).

Examining the detection rates across data sets we see, as expected, that Easy Coins is substantially easier than the other three data sets, and Hard Coins is substantially harder. The 5 cent and 10 cent coins are roughly equally difficult, with LFWF and APWF performing around 20% better on the 10 cent coins than CBF. The 10 cent coins may be harder for CBF because the 10 cent coins are larger than the 5 cent coins, and consequently there is greater area over which individual localisation positions may be spread. We will attempt to examine this effect in the next section.

The FAR and FAA measures also confirm that Easy Coins are much easer while Hard Coins are much more difficult, and as expected the 5 and 10 cent coins have similar difficulty.

Examining Easy Coins, we see that all three fitness functions produce very similar detection rates (within 1% of each other), however LFWF resulted in 5% greater false alarm rate than APWF (0.05 greater number of false alarms per object) and CBF resulted in almost twice as many false alarms as APWF. This trend of CBF producing around twice as many false alarms as LFWF and APWF is repeated in all data sets. APWF produced between $1\frac{1}{2}$ times to 2 times as many false alarms as LFWF on the 5 cent, 10 cent and Hard Coin data sets.

On all but the 10 cent coin problem, all three fitness functions produced detection rates within 2 to 3% of each other. LFWF marginally outperformed APWF and CBF on the 5 and 10 cent coins while APWF marginally outperformed LFWF and CBF on the other two data sets.

The false alarm area per object measurements show the difference of the results for the three methods prior to clustering. On all data sets, CBF performs substantially worse than both LFWF and APWF, producing around 3 to 10 times as many individual localisation positions (ie: false alarm pixels) per object.

We know now that all three fitness functions produce similar detection rates, while CBF tends to produce more false alarms. Examining the training times we see that, as predicted, both new fitness functions require much less training time than the clustering based method. On the 5 cent, 10 cent and Hard Coin data sets, CBF required around four times as much evaluation time as the new fitness functions, with similar number of generations. On the Easy Coins, CBF evolved programs in only 14 generations whereas LFWF and APWF both required around 37 generations, consequently CBF achieved comparable training time on

this instance.

4.9.1 Further Analysis

Table 4.2 described the overall training performance and details the accuracy when evolved programs are applied to the test set. Tables 4.3 and 4.4 examine the individual localisations produced by the different fitness functions. These form initial analysis of the results produced by the fitness functions before clustering has grouped individual localisation positions. Precision (P) and Recall (R) are used to analyse the results instead of Detection Rate and False Alarm Rate, partly to avoid confusion with the use of Detection Rate and False Alarm Rate for the results after clustering, and also because they provide more useful insight. Recall is identical to Detection Rate, while Precision measures the percentage of individual localisation positions which are correct.

Table 4.3 analyses the individual localisation positions which fall within a tolerance of 3 pixels from the true object centres. For example, the first row informs us that when programs are trained on the Easy Coins using LFWF, they tend to find 97.28% of target objects with some individual localisation positions, however only 24.50% of these individual localisations are within the 3 pixel tolerance.

Table 4.4 attempts to examine the individual localisation positions from the point of view of each entire cluster within the bounds of the objects (or within the typical radius of the objects). It does this by considering the individual localisation positions in the same way as table 4.3 but with a tolerance set to the radius of the largest objects. This is also the radius used for the fitness radius and is half the width of the input window. So, it measures the precision as the number of target objects which have any individual localisation positions fall within the 35 pixel tolerance of its true centre as the percentage of the total number of localisation positions produced. The recall is the number of target objects with any individual localisation positions within this tolerance of their centre as a percentage of the total number of target objects. Table 4.4 also shows the positional error (PosError) or the average distance in pixels of these localisation positions from the true object centres. For example, the first row states that LFWF applied to the Easy Coins produced 99.98% of localisation positions being within the 35 pixel tolerance of some object centre and 98.03% of objects were found somewhere within the tolerance of their centre. Also it shows that, on average, the localisation positions where 5.3 pixels from the true centre of objects.

Dataset	Fitness function	Accuracy	
		P (%)	R (%)
Easy Coins	LFWF	24.46	97.28
	APWF	21.03	97.77
	CBF	9.04	100.00
5 cent Coins	LFWF	17.18	83.40
	APWF	13.31	89.58
	CBF	3.21	99.75
10 cent Coins	LFWF	19.96	81.58
	APWF	13.73	93.36
	CBF	3.86	99.98
Hard Coins	LFWF	10.65	59.19
	APWF	8.10	70.65
	CBF	1.82	97.39

Table 4.3: Precision and Recall of Localisation Positions within 3 pixels of Object Centres

Dataset	Fitness function	Accuracy		
		P (%)	R (%)	PosError
Easy Coins	LFWF	99.98	98.03	5.30
	APWF	100.00	99.31	5.78
	CBF	99.89	100.00	8.24
5 cent Coins	LFWF	99.39	87.78	6.55
	APWF	99.25	92.97	7.29
	CBF	99.33	100.00	14.50
10 cent Coins	LFWF	99.43	83.19	6.28
	APWF	99.63	96.88	7.93
	CBF	98.17	100.00	12.67
Hard Coins	LFWF	98.41	72.05	8.69
	APWF	97.81	87.21	9.87
	CBF	93.50	99.94	17.78

Table 4.4: Analysis of Localisation Positions within 35 pixels of Object Centres

The recall informs us if objects have been found at all, so that we can examine why the recall is low for the harder data sets. The precision helps us to gauge how many false alarms are being produced on the background image. If the 35 pixel tolerance based precision is 100% and FAA (from table 4.2 is large, then a great many false alarms are being produced within objects, but none in the background.

From table 4.3 we know that only 24.5% of localisation positions were within 3 pixels of object centres when LFWF was applied to the Easy Coins, and that agrees with what we have just seen about the average positional error. We see that localisation positions are tending to be away from true object centres. In other words, large clusters are being formed. This highlights the need for clustering of results.

Our analysis of detection rate and false alarm rate, after clustering, shows that the three fitness functions achieve similar detection rates on each single data set: around 81%, 54%, 31 to 53% and 28% on each of Easy Coins, 5 cent Coins, 10 cent Coins, and Hard Coins respectively. Table 4.3 informs us that 98%, 83 to 99%, 81 to 99% and 59 to 97%, respectively, of objects have individual localisation positions within three pixels of true object centres, and table 4.4 informs us that around 93 to 100% of all localisation positions are within objects somewhere, ie: not on the background part of the test images. This raises the question of what brings the detection rate down.

4.9.2 Detection Map Analysis

Detection maps are a visual way to see how the object detections map onto the original test images. Figure 4.4 presents a serious of cut-outs from detection maps. Each test data set contained images with a total of 128 objects. The detection maps shown show the cut-out area at 15 objects each, to provide a guide to the full detection map of all 128 objects which would be too large to present here. Typically all three fitness functions were found to produce very few detections on the background and so we can avoid the extra space by ignoring the background section of the test images.

These detection maps represent the typical results when each of the three fitness functions are applied to the 10 cent Coins data set. Each row-wise pair show the unclustered and clustered results for a single fitness function: LFWF, APWF, and CBF in order down the page. (a) shows the unclustered detection map using LFWF, and (b) shows the clustered detection map for the same objects using LFWF also, (c) and (d) show the APWF results,

and (e) and (f) show the CBF results.

The previous two sections raised the questions: (i) why does CBF perform worse on the 10 cent coins, when its detection rate is similar to the others for the other three data sets, and (ii) what causes the detection rate to be so low when it can be seen that 93% of objects are found within the 35 pixel tolerance at worst. Here we can begin to answer these questions.

The detection maps on the right of figure 4.4 use a "cross" to represent a successful detection within three pixels of the object centre, and a "square" to represent a detection which is considered a false alarm because either the object has already been detected or the detection is too far from the true centre. We see that, by eye, all objects appear to be successfully detected, however the squares on some of the objects inform us that many false alarms were very close to the object centres, but have been rejected because they were not close enough.

From the results of LFWF, this appears to be directly related to the size of the clusters. The small clusters produce positionally accurate results, while the results from the larger clusters are rejected as false alarms. This trend is continued to some extent in APWF. CBF, however, has produced very large clusters, most of which have produced false alarm detections. This would seem to be the cause for CBF performing so much worse than LFWF and APWF on the 10 cent Coins.

The results of figure 4.4 show us that we should consider summarising the detection results for different tolerances. The results of localisation will have different constraints depending on the process which uses the detection results and so a plot summarising for multiple tolerances would be very useful when designing later stages or selecting fitness functions to meet the requirements of the later stages. Tables 4.5 and 4.6 present results summarising the detection rates and false alarm rates for all fitness functions on all data sets with a serious of different values for TOLERANCE. Tables such as this are hard to interpret, however a lot of information is present. For the sake of brevity, a single plot has been produced which presents the detection rates of each fitness function averaged over the four data sets. See figure 4.5. This is by no way a perfect description, however it gives a good guide. The false alarm rates can be directly inferred from the values of the detection rates because the increasing tolerance accepts more detections as correct and thus reduces the number rejected as false alarms in a linear fashion.

Examining this plot we see that with greater values of TOLERANCE, a greater detection rate is achieved. All fitness functions achieve their highest detection rate with a TOLERANCE of 10 pixels or greater. We also see that CBF produces better detection rate than LFWF and APWF when the TOLERANCE is set greater. (Note that the tolerance setting here does not affect how the programs were trained, CBF still used a 3 pixel tolerance to calculate its detection rate during training).

4.10 Chapter Conclusions

From our analysis of results we come to the wonderful conclusion that LFWF and APWF produce similar detection rates to CBF on three of the four data sets, and significantly better detection rates on the 10 cent coins. Also, both LFWF and APWF have produced fewer false alarms. However, of significance, is that CBF took around four times as long to evolve its programs for the same number of evolutions as LFWF and APWF. We would not expect LFWF and APWF to produce better detection rates than CBF because our goal was to produce a similar detection rate with much greater training efficiency. Results suggest that our goal has been achieved.

APWF and LFWF performed with very similar detection results on all four datasets,

Dataset	Fitness function	Detection Rates (%) for different TOLERANCES											
		TOLERANCES											
		0	1	2	3	4	5	7	9	10	15	20	30
Easy Coins	LFWF	7.3	27.2	54.3	81.0	92.7	96.4	97.6	97.8	97.9	98.0	98.0	98.0
	APWF	7.5	27.2	55.1	82.1	93.9	97.8	99.2	99.3	99.3	99.3	99.3	99.3
	CBF	5.5	24.4	51.1	81.9	96.5	99.7	100.0	100.0	100.0	100.0	100.0	100.0
5 cent Coins	LFWF	3.0	12.5	30.3	54.5	69.8	79.5	84.4	85.5	85.7	86.8	87.4	87.7
	APWF	2.9	12.4	29.9	54.4	71.7	83.1	90.0	91.3	91.4	92.3	92.7	93.0
	CBF	3.3	13.0	28.3	51.7	69.7	84.5	95.3	98.9	99.5	100.0	100.0	100.0
10 cent Coins	LFWF	3.1	15.0	32.4	52.9	65.4	74.4	80.4	82.1	82.4	83.1	83.2	83.2
	APWF	2.6	13.5	29.0	50.4	65.0	77.3	88.7	93.7	94.9	96.6	96.8	96.9
	CBF	1.5	6.2	15.4	31.1	48.4	67.3	87.7	97.3	98.7	100.0	100.0	100.0
Hard Coins	LFWF	1.4	6.4	15.0	27.3	37.1	45.2	55.1	61.4	62.9	66.0	68.6	70.5
	APWF	1.5	7.0	15.9	29.3	40.7	50.6	63.4	72.5	74.9	80.0	83.1	85.8
	CBF	1.5	5.1	13.6	27.5	40.5	56.4	72.5	84.8	87.3	93.7	96.2	98.7

Table 4.5: Detection Rates for different values of TOLERANCE, averaged over all data sets

Dataset	Fitness function	False Alarm Rates (%) for different TOLERANCES											
		TOLERANCES											
		0	1	2	3	4	5	7	9	10	15	20	30
Easy Coins	LFWF	122.3	102.3	75.3	48.5	36.8	33.2	31.9	31.8	31.7	31.5	31.5	31.5
	APWF	118.0	98.3	70.4	43.4	31.6	27.6	26.3	26.2	26.2	26.2	26.2	26.2
	CBF	168.6	149.7	123.1	92.2	77.6	74.5	74.1	74.1	74.1	74.1	74.1	74.1
5 cent Coins	LFWF	187.3	177.8	160.0	135.8	120.5	110.8	105.9	104.8	104.6	103.5	102.9	102.6
	APWF	208.8	199.3	181.8	157.3	140.0	128.5	121.7	120.4	120.2	119.4	118.9	118.7
	CBF	277.9	268.1	252.9	229.5	211.4	196.6	185.9	182.2	181.6	181.1	181.1	181.1
10 cent Coins	LFWF	113.0	101.1	83.7	63.2	50.7	41.7	35.7	34.0	33.6	32.9	32.9	32.9
	APWF	184.0	173.1	157.6	136.2	121.6	109.3	97.9	92.8	91.7	90.0	89.7	89.7
	CBF	221.6	216.9	207.7	192.0	174.8	155.8	135.4	125.8	124.4	123.2	123.1	123.1
Hard Coins	LFWF	118.8	113.7	105.1	92.8	83.0	74.9	65.0	58.7	57.2	54.1	51.5	49.6
	APWF	213.4	207.9	199.0	185.5	174.2	164.3	151.4	142.4	139.9	134.8	131.8	129.1
	CBF	246.8	243.2	234.6	220.7	207.7	191.9	175.8	163.5	160.9	154.5	152.1	149.6

Table 4.6: False Alarm Rates for different values of TOLERANCE, averaged over all data sets

however LFWF tended to produce either very similar or fewer false alarms. Training times of both new fitness functions were very similar.

CBF is not good at constraining the spread of clusters, as we have seen. It attempts to reduce the size of clusters using FAA and the effect of applying clustering before the detection rate is calculated causes it to attempt to produce clusters which have centres close to object centres. However, the lack of ability to effectively constrain the spread of clusters could be the reason for its poor detection rate on the 10 cent coins. LFWF, however, has a specific way of keeping both the number of false alarm pixels and positional error low, and we have seen this from the results. It appears that this allows LFWF to produce better results than in some cases because it constrains the clusters better. This is certainly highlighted by figure 4.4.

It should be noted here that only one set of CBF parameters (K1 to K3) where used in our experiments. The values used in work by another author were used. It may be that better results could be achieved if different values are used, however this highlights one of the disadvantages that have been found with this method. Namely that the existing fitness functions use "free parameters" which are hard to determine good values for.

4.10.1 Localisation Fitness Radius

The parameters affecting the Localisation Fitness Radius depend very largely on the ability of the classification stage, and not directly on the size of the objects. This is simply because it depends on how flexible the classifier is at accepting cutouts whereby the objects are not always exactly centred.

This is partly defined by the size of the objects trained on, and also on the training techniques. A large number of parameters can affect this radius, and consequently only experimentation and rough heuristics can determine its value. Also note that the radius is really setting a weighting factor of how important this feature is against other parameters to the fitness function. These issues are largely outside the scope of this project.

We do, however, suggest suggest some basic heuristics. The exact value of the fitness radius will not affect the overall effectiveness greatly. It is likely that it is best suited to detecting objects of similar sizes. If the localiser is detecting objects of different sizes, then either the radius of the smallest object or of the largest object could be used. Alternatively the average expected radius could be used.

Our data set contained objects of two separate radiuses: 35 pixels and 29 pixels. We chose the larger of these as our fitness radius.

4.10.2 Complex Objects

The fitness functions we have devised here worked on the basic assumption that we are dealing with round objects. We decided that a sensible heuristic for the value of the Fitness Radius is the radius of the smallest or largest object, or possibly the average radius. The exact value of the fitness radius should not play a major role in the performance of the learning system. Its effect is on the relative importance of positionally correct localisations versus the error of having false alarms. For this reason, it is more a measure of how tolerant the classification stage is to positionally innaccurate localisations, than of the object size.

Further research would need to investigate the effect of the value of the Fitness Radius on the effectiveness of trained programs on test data sets. We suggest that our fitness functions may be applicable to most object localisation problems, not just to round objects as we have done. We have stated that the fitness radius is affected by the flexibility of the classifier, rather than the characteristics of the objects, and therefore it should not be affected by the shape of objects either. So it seems that objects of complex shape may be accommodated by our new fitness functions. Future research will need to address this also.

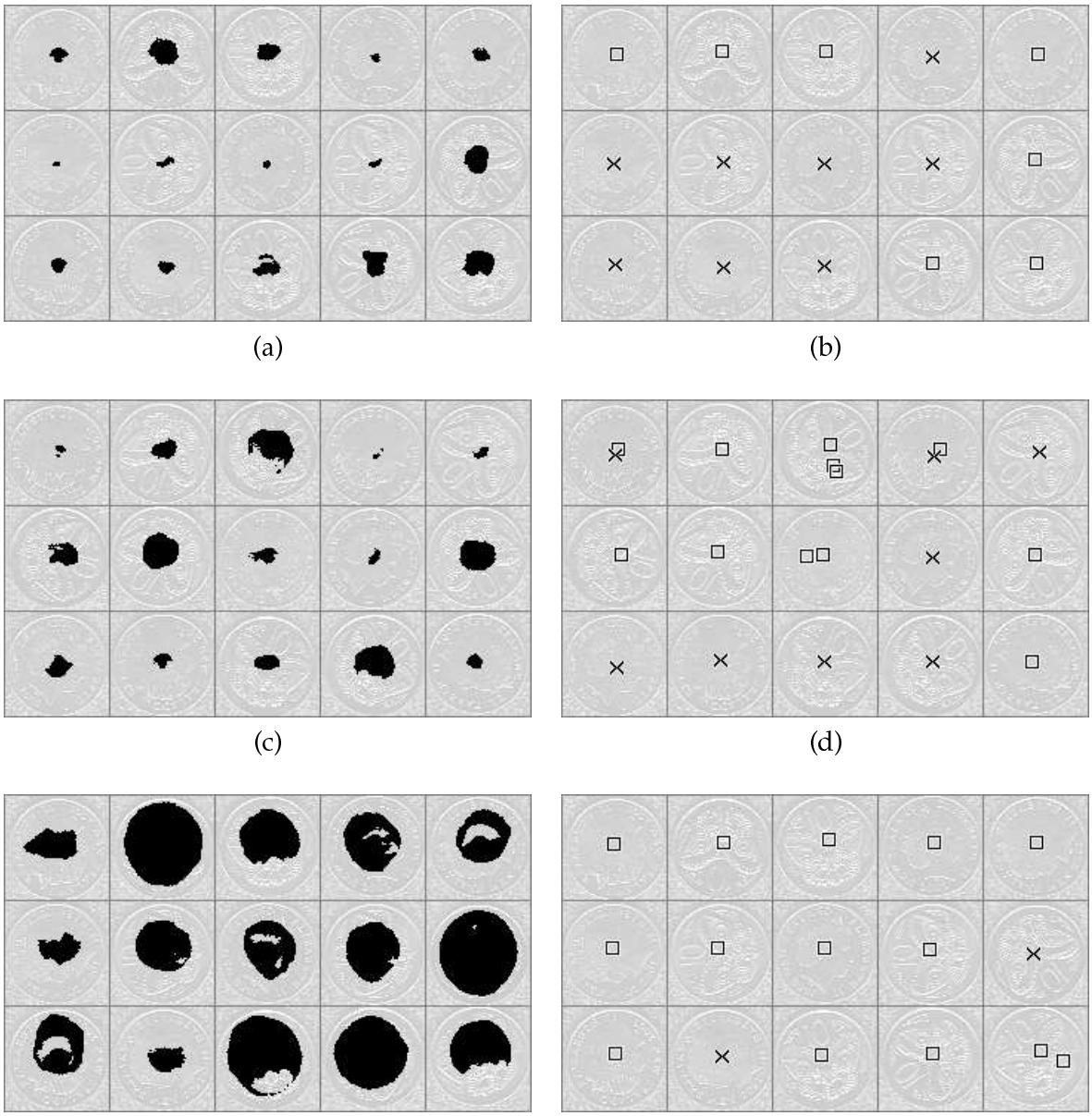


Figure 4.4: Detection maps for (a,c,e) Unclustered Results and (b,d,f) Clustered Results using: (a,b) LFWF, (c,d) APWF, and (e,f) CBF (Crosses are accepted detections, squares are detections which have been rejected as false alarms

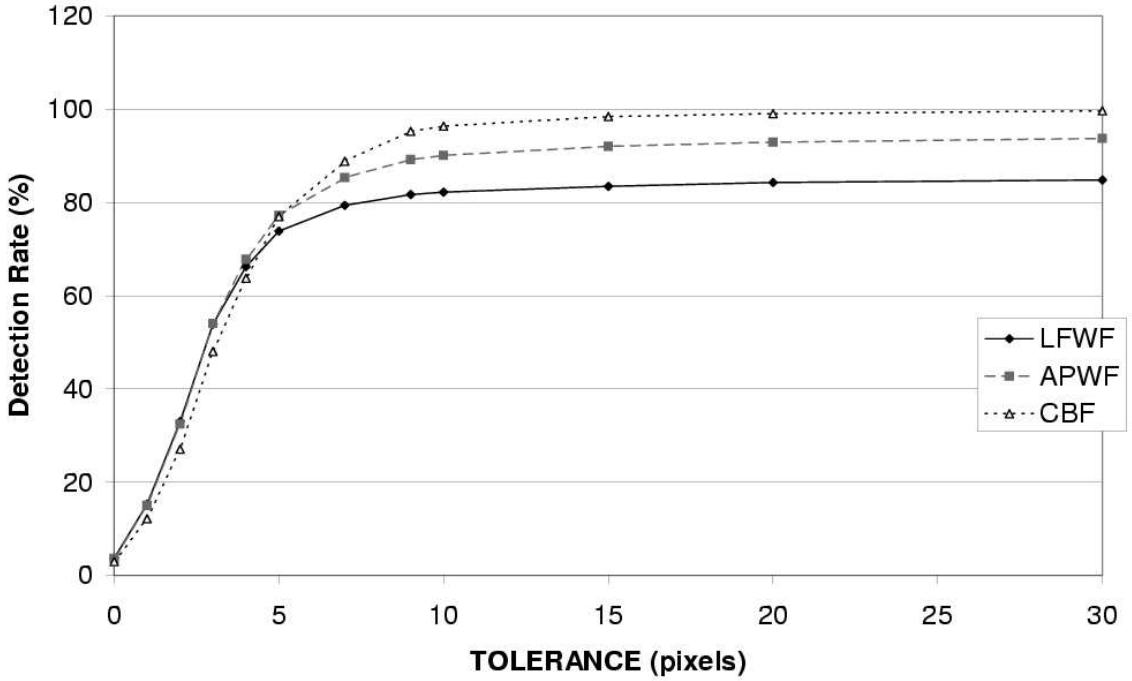


Figure 4.5: Detection Rates for different values of TOLERANCE, averaged over all data sets

Chapter 5

Optimising Training Data

This chapter introduces the concept of *Training Data Proportions* and the types of data which may be used to train with. It then describes in detail some characteristics and assumptions which enable us to reduce the possible number of cases to examine so that we may do experiments to search the relevant parameter space. Finally we describe the experimental results and draw (some very interesting and surprising) conclusions.

5.1 Typical Training Data Preparation

We could train our object detection system with a full set of cut-outs taken from a window which is swept over the training images. However, for large images this creates a vast number of training examples making the training time unsuitably long. One method used is to reduce the total number of training examples by using a combination of hand-chosen and randomly chosen samples from the training images [11, 2, 10]. In this project we seek to answer the question, "are some samples better than others and if so, how do we pick the better ones?".

5.2 Training Data Types

One can consider that there are a number of different types of data which can be used for training localisers, the most obvious and basic of which are "positive" and "negative" examples. But which label do we attach to an example with half object and half background? Our new fitness functions addressed this issue with regards to learning from such data, we now turn our attention to what kinds of training examples we should train with, and, more specifically, what proportions of these different kinds.

We have identified four basic types of training example from our work on the fitness functions. These are described below and illustrated in figure 5.1.

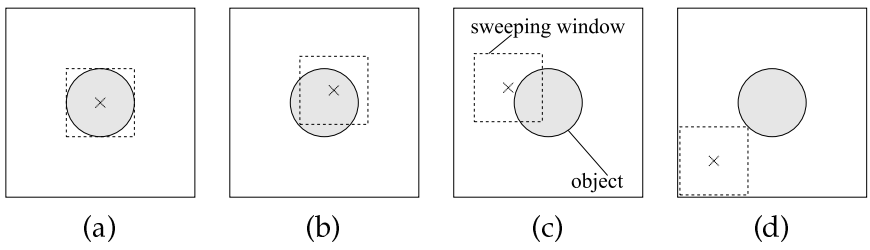


Figure 5.1: Examples of the different types of training data, these are caused by different input window positions.

1. *Exact Centre.*
   Typically there are only a few cases of these. The images we used each contained only 16 objects (and thus 16 exact centre pixels) out of approximately 3 million pixels.

2. *Close to Centre.*
   These training examples have the centre of the input window falling within the bounds of an object.

   This may be defined as needed for objects of irregular shape. We worked with round objects of two different sizes and for simplicity we used the typical radius of the larger objects as an "ideal object radius" and applied this to all objects when determining if the centre of the input window was within the object bounds. This eliminated the need to explicitly define the bounds of each object.

3. *Include Objects.*
   This is defined as all training examples which contain any pixels from any object, but which are not considered "Close to Centre" training examples. Again the "ideal object radius" is used and consequently the smaller objects will result in some of these examples not containing pixels from any objects, however the effect of this error is small.

4. *Background.*
   These are all remaining examples which do not include any pixels from any of the objects of interest. These may also contain objects which we are not interested in, particularly in cluttered images.

5.3 Assumptions and Hypotheses

For a given problem domain within localisation, we assume that there is some proportion of these four types which is optimal in the sense that the best detection results are achieved when the detector is trained using this proportion. It is very likely that these "Training Data Proportions" will be dependent on the domain because, for example, cluttered backgrounds will be harder to distinguish from objects than uncluttered backgrounds. We produce experimental results in the hope that the we may uncover some hidden characteristics and at the very least provide useful guides for problems similar to the ones we have investigated.

With a good proportion chosen, one can vary other parameters, such as training set size, number of genes, or population size, to achieve the desired level of fitness. Research done by Boonyanunta and Zeephongsekul in [4] suggests a useful mathematical model for predicting the improvement in fitness by increased training set size.

If we find, for a given problem domain, that certain proportions of the four types of training example are optimal when training on a small training set, we will assume that these same proportions will be optimal, or at least close to optimal, when we use a larger training set. Indeed this should be true if the data is randomly sampled and the same method is used for both small and large training sets.

5.4 Introduction to Experimental Setup

In our experiments, we first note that there are only a very small number of Exact Centre examples and we thus assume that either: (a) best results are achieved by using all of them, or (b) no significant loss is incurred by doing so. As such, we always include all of the first

type of training example and vary the proportions among the remaining three types in order to find the optimal training data proportions.

The remainder of the training data is constrained by the following:

$$C + I + B = 100\% \tag{5.1}$$

where:

- C = percent Close to Centre.
- I = percent Include Objects.
- B = percent Background.

This has the nice feature that it represents only a single plane in three dimensional space (see figure 5.2), effectively reducing the parameter search space to a two dimensional one.

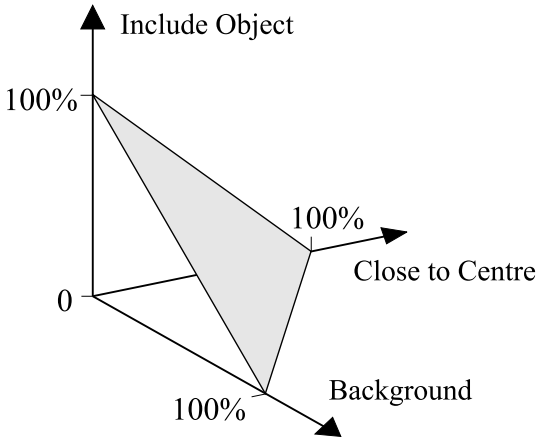


Figure 5.2: Training data proportions: the possible proportions are represented by the surface of the plane.

In our experiments, we used all four data sets as for previous experiments and set the basic parameters as described in table 5.1. We experimented with 28 separate proportions sampled from the plane in figure 5.2 and conforming to equation 5.1 which are listed in table 5.2. Each entry represents the value of I for a given C and B. For example, the first three entries in the first row indicate that, using no background (B = 0), we will try 0% Close to Centre with 100% Include Objects, 17% Close to Centre with 83% Include Objects, and 33% Close to Centre with 67% Include Objects.

For each sampled proportion, 100 experiments were carried out. These were made up of 10 different random seeds when extracting the training data from the source images, by 10 different random seeds for the GP learning system.

The basic genetic parameters and details of training data have been set as listed in table 5.1.

5.5 Results

Table 5.3 lists the results of the LFWF fitness after 50 generations when the different training data proportions are used on each of the four data sets. For example, the first row of (a) shows us that using 0% Close to Centre and Background (and thus 100% Include Objects) gives us 0.00017% fitness, while 17% Close to Centre, 0% Background, and thus 83% Include Objects, gives us 0.01533% fitness. The best fitness is achieved on the easy coins with 100%

number of gens	50
initial population	500
mutation rate	30
elitism rate	10
maximum program depth	6
minimum program depth	3
function set	$\{+, -, \times, \%, \text{if}<0\}$
fitness functions	LFWF, APWF
training set size	3968 (samples taken from first 8 images in data set)
validation set size	3968 (samples taken from second 8 images)
test set	full sweep of last 8 images

Table 5.1: GP parameters used

	I% for each C and B						
$B \backslash C$	0	17	33	50	67	83	100
0	100	83	67	50	33	17	0
17	83	67	50	33	17	0	
33	67	50	33	17	0		
50	50	33	17	0			
67	33	17	0				
83	17	0					
100	0						

Table 5.2: Training data proportions tried

Close to Centre. All fitnesses are very low, however recalling how LFWF is calculated we realise that it considers all individual localisation positions and any of these which represent form false alarm pixels will act heavily against a high fitness.

To show a clear trend, figure 5.3 shows the combined fitness over all four data sets for both the LFWF and APWF fitness functions and scaled to show the fitnesses relative to the best fitness found with that fitness function. The rows coming down to the right (and of equal shading) represent the columns in table 5.2, the rows going up to the right represent the rows in the table, and each bar represents the average relative fitness when using the proportions as specified by the single corresponding entry in the table 5.2. Figure 5.4 summarises the number of generations of training required in a similar way. It shows us immediately that, except for the case of 0% close to centre, the number of training generations are all very similar. Not a great deal can be concluded from the number of generations when 0% close to centre is included in training data as we can see from 5.3 that the fitness of trained programs in this case are very low.

Results from the four data sets show that the higher fitness results are achieved when the majority of training/validation examples are of only two types: "Exact Centre" and "Close to Centre". This seems to indicate that these first two types of example contain the most useful information for training. This may be a feature of the LFWF fitness function which is capable of learning well from data which contains both object and background. In fact, this seems to eliminate the need to use background examples at all.

Consider how weighted precision and recall are calculated. Close To Centre examples can be classified as positive (object) with a weight to indicate its correctness. However, if the localiser classifies it as negative (background), this is also given a good fitness because it conforms to the final intent of the learning strategy.

(a)

	Fitness on Test Set (%)						
$B \backslash C$	0	17	33	50	67	83	100
0	0.00017	0.01533	0.03174	0.04813	0.07218	0.09334	0.11997
17	0.00018	0.01435	0.03548	0.05302	0.06355	0.10082	
33	0.00015	0.01576	0.03578	0.04765	0.06531		
50	0.00014	0.01349	0.03352	0.05229			
67	0.00016	0.01290	0.02818				
83	0.00011	0.01416					
100	0.00001						

(b)

	Fitness on Test Set (%)						
$B \backslash C$	0	17	33	50	67	83	100
0	0.00008	0.00990	0.01701	0.02209	0.03022	0.03469	0.03880
17	0.00009	0.01017	0.01688	0.02291	0.02894	0.03499	
33	0.00008	0.01062	0.01707	0.02218	0.03007		
50	0.00007	0.01003	0.01635	0.02477			
67	0.00005	0.00918	0.01992				
83	0.00005	0.00952					
100	0.00003						

(c)

	Fitness on Test Set (%)						
$B \backslash C$	0	17	33	50	67	83	100
0	0.00017	0.01533	0.03174	0.04813	0.07218	0.09334	0.11997
17	0.00018	0.01435	0.03548	0.05302	0.06355	0.10082	
33	0.00015	0.01576	0.03578	0.04765	0.06531		
50	0.00014	0.01349	0.03352	0.05229			
67	0.00016	0.01290	0.02818				
83	0.00011	0.01416					
100	0.00001						

(d)

	Fitness on Test Set (%)						
$B \backslash C$	0	17	33	50	67	83	100
0	0.00009	0.00552	0.00952	0.01655	0.01920	0.02276	0.02954
17	0.00008	0.00535	0.01103	0.01459	0.02054	0.02545	
33	0.00008	0.00498	0.01210	0.01715	0.02037		
50	0.00008	0.00544	0.01003	0.01491			
67	0.00006	0.00460	0.00992				
83	0.00006	0.00534					
100	0.00002						

Table 5.3: Fitnesses when LFWF is applied to Test data sets using different Training Data Proportions on: (a) Easy Coins, (b) 5 cent Coins, (c) 10 cent Coins, (d) Hard Coins data sets.

5.6 Chapter Conclusions

Our results have suggested a most interesting phenomenon: that the best fitness has been achieved when the majority of training examples are of the first two training data types. The plots in figure 5.3 indicate that there is an almost direct linear relation between the final test fitness and the percentage of the training data which is composed of *close to centre* examples. For each value of Close To Centre, the fitness stays almost constant while only the ratio of *background* to *include objects* examples is varied, particularly for LFWF.

The experiments performed used training and validation sets of equal size and of equal training data proportions. Thus it is quite obvious that very bad test fitness should result from training data containing mostly background. If the evolutionary process sees only background data then it will not evolve successful object detectors. At the other extreme, if the evolutionary process does not see examples of background, it may not train good object

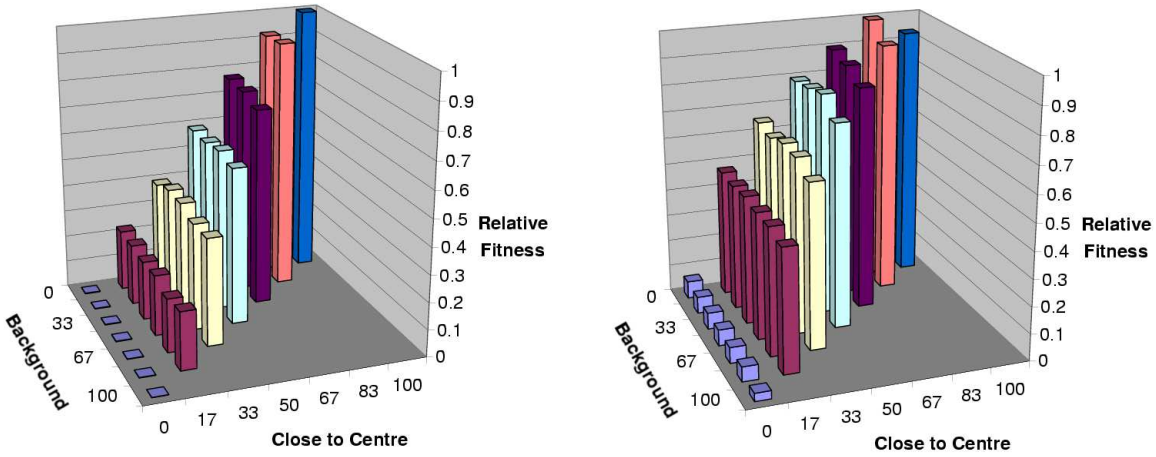
detectors with few false alarms. But the fitness measures we used represent both detection rates and false alarm rates and show us that detection rates are maximised and false alarm rates minimised by increasing the number of training examples of objects.

There may be another aspect affecting our results. In the harder three of our four data sets, the background is noisy but relatively uniform in its level of noise. It is also quite different to the objects being detected. The consequence of this is that programs may evolve very quickly to reject background and only require a few training examples. Accurately detecting object centres, on the other hand is much harder than rejecting background in our data sets.

There are thus two things which future work needs to investigate regarding the training data proportion results:

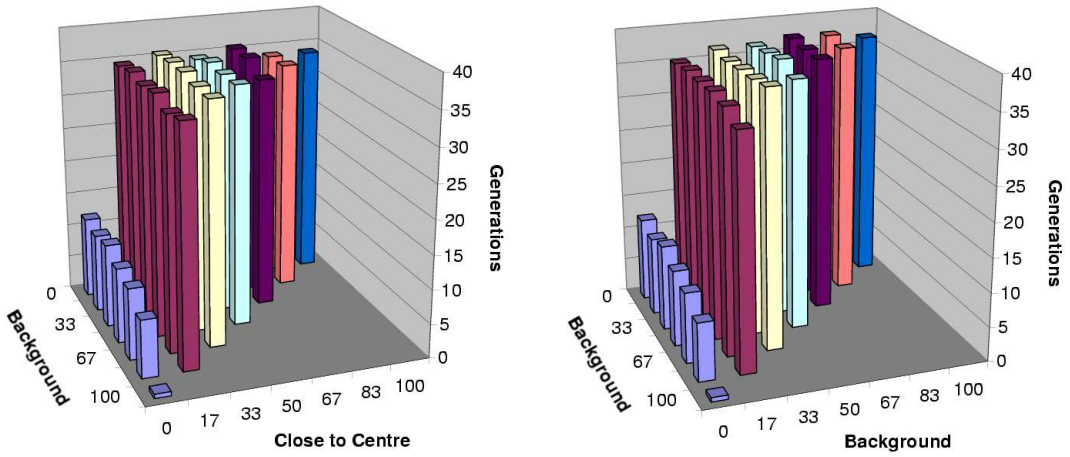
1. What results do we get for data sets where the background is highly cluttered and similar to actual objects. If the background is much harder to reject as non-object, do we need a greater proportion of background examples?

2. What is the best proportions and size for the validation set? The validation set is evaluated only once every evolution, not once for every individual in the population. Consequently it can be much bigger, or even perform an entire sweep of the validation set images, rather than a sampled set of examples.



(a) (b)

Figure 5.3: Relative Fitness using different Training Data Proportions: (a) LFWF fitnesses, (b) APWF fitnesses



(a) (b)

Figure 5.4: Generations to train using different Training Data Proportions: (a) LFWF generations, (b) APWF generations

Chapter 6

Summary

In this project we set out to address some of the problems of fitness functions and training performance for object localisation using Genetic Programming. We applied our methods to similar data sets of increasing difficulty.

We developed two new fitness functions from a careful examination of the combinations of possible localisations of objects and of how their corresponding fitnesses should compare.

LFWF (Localisation Fitness Weighted 'F' measure) was based directly from this study, while APWF (Average Position Weighted 'F' measure) attempted to match more directly the way that clustering algorithms calculate the final position of objects.

These two fitness functions attempted to produce similar localisation effectiveness as clustering based fitness functions without the need of a computationally expensive clustering algorithm during evolution. This type of fitness function has a major advantage because it takes much less time to evolve and thus reduces training time.

We also examined the effect of the proportions of different types of training example on the effectiveness of the evolved localiser. This has implications on the training performance because GP takes longer to train with greater number of training examples and so it is undesirable to train with unnecessary or less important data.

6.1 New Fitness Functions

The new fitness functions where found to produce similar detection accuracy to the clustering based fitness function while taking substantially less computation time for fitness calculations. They where also found to produce around half as many false alarms as the clustering based method.

A detailed analysis of the characteristics of the new fitness functions and of the existing fitness function have highlighted the improvements that this new method has introduced. Our analysis found that programs which produced large clusters of individual localisation positions tended to produce low detection rate because only the smaller clusters tended to be detect objects to the desired tolerance level. Our results also showed that the clustering based method produced very large clusters and consequently its detection rate was low on occasion.

6.2 Training Data Proportions

The examination of training data preparation highlighted two closely related types of training example which produce the best fitness of evolved programs. We made the assumption that all training examples of exact object centres should be included. It was found that best

fitness was achieved when the remainder of training data contained mostly (around 85 to 100% depending on which fitness function was used) examples of sweeping window positions which are centred within the bounds of objects and thus at least around half of their input pixels are made up from object pixels. This is considered to be a very useful and interesting discovery for object detection problems using training methods.

Care must be taken with problem domains which differ greatly from the data sets used by this report as the results found here may not apply to images with backgrounds which are more similar to the objects than in our data sets.

6.3 Free Parameters In Fitness Functions

One disadvantage with the existing fitness functions has been their use of "free parameters" which define the relative importance of the different goals for the fitness function. While it is useful and important to have that control, it can be a great hindrance when first starting. Furthermore, the values of these free parameters have no specified range within which typical values should be found and the typical values are often orders of magnitude different to each other.

We now discuss the new fitness functions and what parameters they have. The immediately obvious parameter is the Fitness Radius, and this has been discussed at length in previous sections. The combination of precision and recall within the formula for the 'F' measure also creates an implicit parameter which defines the relative importance of detection rate (recall) and detection correctness (precision). It inherently weights the importance of achieving high recall to be equal with the importance of reducing the number of false alarms.

Precision and recall can be given different relative importance, if wished, by modifying the 'F' measure equation. So it is possible to control the relative importance, however there is a direct and obvious default which makes using these methods very easy.

6.4 Future Work

The analysis on the new fitness functions functions highlighted a number of points to be considered for future work. Among them are the following:

1. What is the effect of the Fitness Radius on the localiser effectiveness? Does bad selection of Fitness Radius have a substantial adverse affect on detection rate or false alarm rate?

2. It appears that LFWF and APWF may be applicable to objects of complex shape. This needs to be investigated.

Other suggested future work is discussed in the remaining sections.

6.4.1 Extensions for LFWF and APWF

APWF only measures the positional accuracy of the centre of clusters. It does not measure the extent of the size of clusters, such as using False Alarm Area. It may be possible to combine some measure of false alarm area in a natural way similar to the way that it is included in LFWF, and without the need for free parameters.

Also, neither APWF nor LFWF are restricted to single class localisation problems using a two stage detection method. Both can be extended to calculate the classification accuracy

if they are intended to be used for a single stage detection method. One simple approach to this would be to calculate the weighted precision and recall for each class individually and combine these by taking the average precision and recall before calculating the final fitness.

6.4.2 Further testing of New Fitness Functions

The new fitness functions have been tested and compared to an existing clustering based method by running experiments with up to 50 generations. This allowed us to gauge performance and effectiveness of these fitness functions under difficult circumstances. Namely that they are given a short period of time to train good object detection programs. We saw that under these circumstances, the new fitness functions performed similarly well or better in most cases. These experiments, however, do not test the full ability of the fitness functions to evolve perfect programs. Other work, [2, 11] have typically trained for between 100 to 200 generations in order to achieve good results. Similar experiments should be done and the same analysis performed. It may be found that the new fitness functions converge quickly to their best results, but that methods such as CBF eventually train far better programs for object detection. Or, LFWF and APWF may still perform better for similar problems.

6.4.3 Further Training Data Proportions Experimentation

Future work needs to continue to experiments started for Training Data Proportions by experimenting with different data sets containing highly cluttered background which is similar to the objects of interest.

A more complete survey of theories of validation set size versus the training set size would also be useful. And then to perform further tests experimentation with the effect of the validation size on the final fitness and how this interrelates with the training data proportions.

6.5 Single Stage vs. Multiple Stage Detection Methods

The work presented in this report focuses on the use of a two stage object detection system. The first performs localisation to find the positions of objects at a high positional accuracy. The second classifies all objects at the positions found by the first stage. This is different to previous multiple stage detection methods which used many different methods that are often domain specific.

LFWF may produce programs with poor *positional accuracy* and one possible solution may be to use a final localisation stage, after classification, to find the exact positions of objects, and perhaps even more detailed information such as the coordinates of their outline, or some other indication of rotation, shape, etc. The advantage of performing such a final localisation *after* classification is that the knowledge of the object's class can be used as a guide for the final localiser. For example, one localiser could be trained to localise a single class, with one localiser for each class. These class specific localisers would likely have much better positional accuracy than a generic multi-class localiser.

One could imagine a system which involves four stages:

stage 1: Initial "there is something around here" localisation using a quick scan of an image, producing only a region in which an object of interest may reside, rather than a single (x,y) coordinate. This has an advantage that if a great many number of images need processing, this can quickly discard useless images. (Later reports will describe this in more detail). These localisers are trained to identify a sweeping window position as containing an object if there are any object pixels contained within it.

stage 2: localisation. Starting with the areas declared as "something within this region", find all object centres to within some predefined positional accuracy. The positional accuracy does not need to be very good during this stage. With reduced constraints on the positional accuracy, the emphasis can be put into gaining high recall.

stage 3: classification. Use the localised object cutouts from stage 2. Use the cutout size sufficient to handle the maximum tolerated *positional error* so that it is guaranteed the entire object is contained within the cut-out for classification.

stage 4: For each object found and classified, use the final localiser which has been tailored to the object's class to determine its exact centre and any other positional or rotational information needed.

Advantages This provides a step-by-step style, and thus can be stopped early if no useful information is being extracted. Such as no objects of interest found after completion of the first stage. It is likely that the final localiser would be very accurate under this scheme.

Disadvantages This requires more training time than a system using only a one or two stage detection, because there are more stages to train programs for. The classifier also will be harder to train because it needs to cope with some level of positional error, whereas the simpler systems assume no positional error on the object cutouts passed to the classification stage.

Future work needs to determine whether the cost of the extra training times for having more stages is less or greater than the cost of a single more direct and all-encompassing program, for the same effectiveness and accuracy.

Bibliography

[1] BAUM, E. B., AND HAUSSLER, D. What size net gives valid generalization? *Neural Computation 1* (1990), 151–160.

[2] BHOWAN, U. A domain independent approach to multi-class object detection using genetic programming. Tech. rep., School of Mathematical and Computing Sciences, VUW, 2003.

[3] BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D., AND WARMUTH, M. Learnability and the vapnik-chervonenkis dimension. In *Proceedings of the 1988 Workshop on Computational Learning Theory* (San Mateo, CA, 1989), Morgan Kaufmann.

[4] BOONYANUNTA, N., AND ZEEPHONGSEKUL, P. Predicting the relationship between the size of training sample and the predictive power of classifiers. *???* (2004).

[5] CHOW, R. Multiple class object detection using pixel statistics in neural networks. Tech. rep., School of Mathematical and Computing Sciences, VUW, 2002.

[6] DE GARIS, H. Genetic programming: Modular evolution for darwin machines. In *Proceedings of the 1990 International Joint Conference on Neural Networks* (1990), L. Erlbaum, Ed., pp. 194–197.

[7] DEBEVEC, P. A neural network for facial feature location. UC Berkeley CS283, Project Report, http://www.debevec.org/FaceRecognition/, December 1992.

[8] KOZA, J. R. *Genetic Programming: on the programming of computers by means of natural selection.* Cambridge, MA: The MIT Press, 1992.

[9] MONTANA, D. J. Strongly typed genetic programming. BBN Technical Report 7866, Cambridge, MA 02138, March 1994.

[10] NY, B. Multi-class object classification and detection using neural networks. Tech. rep., School of Mathematical and Computing Sciences, VUW, 2003.

[11] PRITCHARD, M. Genetic programming for multi-class object detection. Tech. rep., School of Mathematical and Computing Sciences, VUW, 2002.

[12] SCHNEIDERMAN, H., AND KANADE, T. Object detection using the statistics of parts. *Int. J. Comput. Vision 56*, 3 (2004), 151–177.

[13] SMART, W. Genetic programming for multi-class object classification. Tech. rep., School of Mathematical and Computing Sciences, VUW, 2003.

[14] SMART, W. Multiclass object classification using genetic programming. Tech. Rep. CS-TR-04-2, School of Mathematical and Computing Sciences, VUW, Feb 2004.

[15] SMART, W. Probability based genetic programming for multiclass object classification. Tech. Rep. CS-TR-04-7, School of Mathematical and Computing Sciences, VUW, July 2004.

[16] WASSERMAN, P. D. *Advanced Methods in Neural Computing*. Van Nostrand Reinhold, New York, 1993, ch. 11. ISBN: 0-442-00461-3.

[17] WIDROW, B. Adaline and madaline – 1963. In *IEEE 1st Int. Con. on Neural Networks* (San Diego, CA, 1987), vol. 1, pp. 143–158.

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

[19] ZHANG, M., AND CIESIELSKI, V. A domain independent approach to multiclass 2d object detection using neural networks and genetic algorithms. Tech. Rep. CS-TR-02-2, School of Mathematical and Computing Sciences, VUW, Feb 2002.

[20] ZHANG, M., AND CIESIELSKI, V. Neural networks and genetic algorithms for domain independent multiclass object detection. *International Journal on Computational Intelligence and Applications 4*, 1 (2004), pp. 77–108.

[21] ZHANG, M., CIESIELSKI, V., AND ANDREAE, P. A domain independent approach to multi-class object detection using genetic programming. Tech. Rep. CS-TR-02-4, School of Mathematical and Computing Sciences, VUW, Feb 2004.

Appendix A

Weighted Precision and Recall

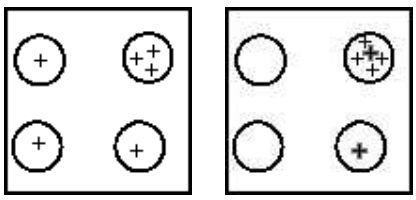
Objects may be localised more than once, in different locations. For training purposes, it is useful to detect this in a very particular manner. At first it seemed that multiple localisations of a single object needed to be treated as a single correct localisation (the "best" one), and the rest as false extra localisations. It later was realised that the distance of these extra localisations from the object centres needed to be considered also. We present here a description of reasoning considering our initial conclusion because it is clearer to explain. We then make a small adjustment to consider the positional accuracy of all localisations.

Remember equations 4.2 and 4.3 for weighted precision and recall, and abbreviate this to:

$$precision = \frac{S}{L}$$
$$recall = \frac{S}{N}$$

where:

- S is $\sum_{i=1}^{N} localisationFitness_i$
- L is number of localisations made
- N is the actual number of objects which should be positively classified.



(a) (b)

Figure A.1: Object localisation examples

Let us consider three possible ways of calculating "positive" precision and recall.

Example 1 Accept all "correct" localisations individually:
In this case, for the example in A.1(a), S ∼ 8, N = 4, L = 8. And precision ∼100% (8/8), and recall ∼200% (8/4).

Multiple localisations of objects cannot be considered separately, or the fitness function would act as if there are more objects than truly present (recall > 100).

Example 2 Accept only best localisation for each object, ignore all others in that object: Could pick only the best localisation for each object, such as indicated in figure A.1(b) by the bold crosses, however, here, S $\sim$ 2, N = 4, L = 2. This gives precision $\sim$100% (2/2) and recall $\sim$50% (2/4).

However, no distinction has been made between the example of the top right hand object and the bottom right hand object. Programs which produce only single localisations should be considered more fit than programs which produce multiple localisations.

Example 3 Accept only best localisation for each object in sum, but still count all localisations made:
Instead, count all localisations in C, and pick the best fit localisation for inclusion in S. Thus example 2 gives: S $\sim$ 2, N = 4, L = 6. And, precision $\sim$33% (2/6), and recall $\sim$50% (2/4).

So, when localising, the fitness of each localisation per object is stored, and the highest fitness value included in the sum of correct localisations. The total number of localisations made, L, is the exact number of localisations made, regardless of their individual localisation fitnesses.

The third option is the option chosen, it's formula is as follows:

$$\mathit{localisationFitness}_i = \sum_{i=1}^{N} \mathit{max}(\mathit{localisationFitness}_{j,i}, j = 1..L_i) \tag{A.1}$$

where:

- N is number of objects.
- $\mathit{localisationFitness}_{j,i}$ is j-th localisation fitness of object i.
- L_i is number of localisations made to object i.

This has the following characteristics:

- It prefers few localisations per object.
- Prefers localisations closer to the object centres.
- Prefers more objects to be localised.
- Prefers more localisations to be correct (not background).
- Some number of extra localisations, say 'x' extras, is considered the same fitness as 'x' localisations of background (false positives).
- The weights of extra localisations (other than the best localisation), for each object, don't affect the sum of correct localisations, they only affect the number of localisations made and thus reduce the precision. Consequently, extra localisations close to the object centre are considered just as unfit as extra localisations further from the object centre.

The last characteristic here is a problem. We want the localisations to form a clean cluster around the region of the centre of each object, without outlying localisations which may not be grouped into the main cluster during clustering. This function does not give preference to either. However, we can achieve the desired affect with only a small adjustment. Instead

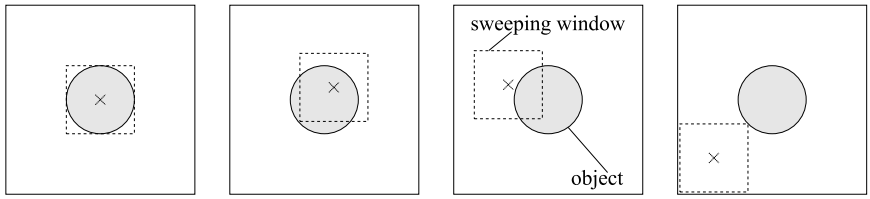


Figure A.2: Examples of different types of training example: (a) should be classified as positive, (b) may be classified as positive or negative, and (c) and (d) should be classified as negative.

of choosing the maximum localisation fitness to represent the fitness with regards to each object, we can average each individual localisation fitness across all localisations for a single object and use that value for that object. Thus we have:

$$localisationFitness_i = \sum_{i=1}^{N} ave(localisationFitness_{j,i}, j = 1..L_i) \tag{A.2}$$

If only a few localisations exist very close to the object centre, then the value for the average localisation fitness will be close to 1.0. If however, the same number of localisations are found for a single object, but they tend to be further from the object's centre, then the value will be less than 1.0. This gives the desired effect.

A.1 Analysing Genetic Program Results

Chapter 5 described four different kinds of training example, these are repeated here in figure A.2 for convenience. We consider that localisation is really classification of the image at specific window positions, and consider how different classifications should be treaded: correct or incorrect? The first kind should always be classified as positive, and a correct classification of this kind receives a weight of 1.0. The example in (b) could be classified as positive or negative. If classified as positive, then it is given some weight less than 1.0, unless that particular object had already been localised, in which case this new localisation will be considered incorrect and the precision will reduce. If, however, (b) was classified as negative, then it would also be considered correct, unless no localisations are made within the bounds of this object, in which case the recall is reduced. Example (c) and (d) should always be classified as negative, any positive classifications are treaded as incorrect and reduce the precision.

Appendix B

Derivation of 'F' w.r.t. True/False Positive/Negative

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$\begin{aligned} F &= \frac{2 \times Precision \times Recall}{Precision + Recall} \\ &= \frac{2 \times \frac{TP}{TP+FP} \times \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}} \\ &= \frac{2 \times \frac{TP^2}{(TP+FP)(TP+FN)}}{\frac{TP(TP+FN)+TP(TP+FP)}{(TP+FP)(TP+FN)}} \\ &= \frac{2 \times TP^2}{TP((TP+FN)+(TP+FP))} \\ &= \frac{2TP}{2TP+FN+FP} \end{aligned}$$

Appendix C

Pattern Files

The pattern file stores the training and test examples which are used to find program fitnesses. The typical format contains only lines of input values and the output class label. The new fitness functions discussed in this report required the pattern file format to be augmented with additional information. This enabled the calculation of the fitness to be efficient. Each training example entry contains a minimum of the following:

1. x, y position within the source image that the example is taken from
2. x, y unique identifier to the nearest object if this example is taken within the radius of some object, plus some way of efficiently finding the x, y position of that object.
3. the desired classification
4. a value indicating the correctness of classifying this example as positive (the localisation fitness)