

Co-operative Co-evolution of Image Feature Extractors and Object Detection Algorithms

Mark E. Roberts

A thesis submitted
to The University of Birmingham
for the degree of Doctor of Philosophy

School of Computer Science
The University of Birmingham
Birmingham B15 2TT
United Kingdom

February 2006

Abstract

Machine learning and pattern recognition problems are commonly decomposed into two stages – feature extraction and classification. The feature extraction stage is often manually designed using knowledge of the problem domain. This results in biases and inflexibility especially in areas such as image analysis where the raw data is large and complex.

This thesis presents a novel framework for simultaneously co-evolving both the feature extraction stage and the classification stage. A representation is defined which allows the evolution of statistical image features. The classifiers, evolved using genetic programming, take the form of programs which manipulate the extracted features. The architecture of the co-evolutionary system is defined along with the underlying collaboration and credit assignment mechanisms, which allow the two stages to implicitly co-operate.

The system is evaluated on synthetic datasets which exhibit noise and variation in scale and orientation, and on complex natural images. Two different training methods are used. In the first, traditional method, every pixel in the image is used for training. Although successful, this method suffers from high computation costs and problems with scaling to more complex problems. In the second method, a multi-stage sampling scheme is used which requires substantially less computation. This method demonstrates similar or better performance than the previous method and is more effective at solving complex problems. In both methods, the co-evolutionary approach is shown to produce solutions that are significantly better than when using a fixed set of features

The success of the approach developed in this work shows that the co-evolution of the two-stage learning processes is possible in this domain, a finding which may have implications in other areas of artificial intelligence and pattern recognition.

Acknowledgements

Firstly I would like to thank my supervisor, Ela Claridge, for her support over the last five years. Despite many other commitments, she always made time to see me and was always inspiring, helpful, and patient during my many topic changes.

I would also like to thank the members of my thesis group, Jeremy Wyatt and Xin Yao, who, despite the short time I often gave them to read my reports, have always given me insightful comments that have no doubt made this a stronger thesis.

There are so many other people at Birmingham that have helped in one way or another that it would be impossible to name them all. A special mention should go to Gavin Brown, with whom I've travelled much of this journey with, and who has gone from being my first year undergraduate C++ tutor, to become a friend and then officemate, and then to be the best man at my wedding. Many others deserve a mention for helping me thrash out ideas on the whiteboard including John Woodward, Dave Brooks, Dave Gurnell, Rachel Harris, Dean Petters, Rob Goldsmith and Noel Welsh.

I'd like to thank Daniel Howard for several useful conversations that helped me to re-implement some of his work which played a large part in Chapter 6.

I gratefully acknowledge Selkirk Remote Sensing Ltd. for allowing me to reproduce some of their imagery.

I want to thank my parents for giving me the start in life that enabled me to get onto this path, but finally, and most importantly, I'd like to thank my wife, Judy, for all of her love and support, and for always being patient and understanding when this work meant that I couldn't give her my full attention. I hereby promise to spend a lot more time with her from now on.

Contents

1	Introduction	1
1.1	Examples	1
1.2	Motivation	2
1.3	Thesis Goals	4
1.4	Thesis Structure	5
1.5	Publications Arising from This Work	6
2	Background	8
2.1	Image Analysis	8
2.2	Evolutionary Computation	10
2.2.1	Genetic Programming	12
2.2.2	Co-evolution	19
2.2.3	Competitive Co-evolution	20
2.2.4	Co-operative Co-evolution	21
2.3	Classification Pipeline	22
2.4	Feature Optimisation	23
2.4.1	Feature Subset Selection	24
2.4.2	Feature Extraction	24
2.4.3	Feature Construction	25
2.5	Evolutionary Computation for Image Processing and Analysis	27
2.5.1	Overview	27
2.5.2	Detection and Classification Systems	28
2.5.3	Geometric Property Classifiers	31

2.5.4	Pixel Statistic Based Classifiers	32
2.5.5	Evolution of Features	41
2.6	Chapter Summary	41
3	Co-evolution of Features and Classifiers	43
3.1	Overview of Previous Approaches	43
3.2	Evolving Feature Construction	44
3.3	Combined Approaches	44
3.4	Co-evolution – A Possible Solution	47
3.5	Chapter Summary	49
4	Co-evolutionary Framework	50
4.1	Representation of Feature Extractors	50
4.1.1	Basic Feature Properties	51
4.1.2	Zone Presentation	53
4.1.3	Evolutionary Operators	55
4.1.4	Evaluation of Rectangular Statistics	56
4.2	Architecture of System	58
4.2.1	Overall Structure	59
4.2.2	Assembling Complete Solutions	61
4.3	Collaborator selection issues	61
4.3.1	Selection Method	61
4.3.2	Credit assignment	62
4.4	Evolutionary Process Summary	63
4.5	Detailed System Description	64
4.5.1	GP Issues	64
4.5.2	Parameters	65
4.6	Chapter Summary	67
5	Using the Framework – Full Image Approach	68
5.1	Method Overview	68
5.2	Fitness function	69

5.2.1	Motivation	69
5.2.2	Implementation	71
5.2.3	Penalty Function	73
5.2.4	A Non-Linear Distance Function	74
5.3	Figure of Merit (FOM)	75
5.4	Experimental Method	75
5.4.1	Overview	75
5.4.2	Function and Terminal Sets	77
5.5	Datasets	77
5.5.1	Synthetic Images	77
5.5.2	Real World Images	81
5.6	Experiments	83
5.6.1	Experiments on Set-T	84
5.6.2	Experiments on Set-TN	86
5.6.3	Experiments on Set-TCN	86
5.6.4	Experiments on Set-TCNS	86
5.6.5	Experiments on Set-TCNR	89
5.6.6	Experiments on Set-TCNSR	89
5.6.7	Experiments on Set-KC	93
5.6.8	Experiments on Set-P	95
5.6.9	Experiments on Set-AT	97
5.7	Analysis	97
5.8	Comparative Analysis	101
5.9	Limitations	103
5.10	Summary	104
6	Using the Framework – A Multistage Approach	106
6.1	Principle of the Multi-Stage Method	106
6.2	Experimental Method	108
6.3	Non-Target Point Sample Selection	110
6.3.1	Problems With Uniform Selection	111

6.3.2	A Non-Uniform Method	112
6.4	Basic Object Detection Experiments	115
6.4.1	Experiments on Set-T	115
6.4.2	Experiments on Set-TN	115
6.4.3	Experiments on Set-TCN	117
6.4.4	Experiments on Set-TCNR	118
6.4.5	Experiments on Set-TCNS	118
6.4.6	Experiments on Set-TCNSR	121
6.4.7	Experiments on Set-KC	121
6.4.8	Experiments on Set-P	125
6.4.9	Experiments on Set-AT	125
6.5	Further Experiments	127
6.5.1	Solution Size and Performance	127
6.5.2	Effect of Feature Set Size	140
6.5.3	Comparison to Fixed Statistic Method	141
6.6	Summary	143
7	Conclusions	144
7.1	Summary	144
7.2	Contributions	146
7.2.1	Contributions of the System	149
7.3	Limitations	150
7.4	Future Work	151
7.4.1	Dynamic Number of Features	151
7.4.2	Generic Features	152
7.4.3	A Generic Co-evolutionary Preprocessor	153
7.5	Concluding Remarks	154

List of Figures

1.1	A car detection task.	3
2.1	A example of a traditional GP tree	14
2.2	Demonstration of crossover	17
2.3	Demonstration of mutation	18
2.4	Antenna designs evolved in [68] and [67]	19
2.5	Traditional classification pipeline	22
2.6	Diagrammatic representations of 4 feature set transformation methods . . .	26
2.7	Feature set used in [124]	34
2.8	Graphical output of feature extractors.	35
2.9	Greyscale detector output	36
2.10	Thresholded detector output	37
2.11	Feature set used by Winkeler et al. [120].	38
2.12	Feature set used by Howard et. al. [46].	39
3.1	Comparison of the traditional and co-evolutionary classification pipelines .	48
4.1	The general form of a feature extraction zone.	52
4.2	Examples of feature zones	54
4.3	Integral image concept	57
4.4	Integral image calculation	58
4.5	Co-evolutionary framework	60
5.1	Discrepancy in naïve fitness function	70
5.2	Example of proximity and distance maps	72

5.3	Example log weighted distance function	74
5.4	Two examples from the set-T dataset.	78
5.5	Two examples from the set-TN dataset.	79
5.6	Two examples from the set-TCN dataset.	79
5.7	Two examples from the set-TCNS dataset.	80
5.8	Two examples from the set-TCNR dataset.	81
5.9	Two examples from the set-TCNSR dataset.	81
5.10	Two examples from the set-KC dataset.	82
5.11	Two examples from the set-P dataset.	83
5.12	Two examples from the Set-AT dataset.	84
5.13	Solution and sample results for set-T	85
5.14	Solution and sample results for set-TN	87
5.15	Solution and sample results for set-TCN	88
5.16	Solution and sample results for set-TCNS	90
5.17	Solution and sample results for set-TCNR	91
5.18	Example of Set-TCNR error	92
5.19	Solution and sample results for set-TCNSR	94
5.20	Solution and sample results for set-KC	96
5.21	Solution and sample results for set-P	98
5.22	Simplified Set-P solution	99
5.23	Solution and sample results for set-AT	100
6.1	Uniform non-target point selection	111
6.2	Non-uniform non-target point selection	114
6.3	Solution and sample outputs for Set-TN	117
6.4	A complete solution for the Set-TCN problem	119
6.5	Primary stage solution for Set-TCNR	120
6.6	A complete solution for the Set-TCNS problem.	122
6.7	A complete solution for the Set-TCNSR problem.	123
6.8	Sample output from the set-TCNSR solution shown in Figure 6.7	124
6.9	Best solution (from 20 runs) for the Set-KC problem.	126

6.10	Sample output from the Set-KC solution	127
6.11	Complete solution for Set-P	128
6.12	Example output from Set-P solution	129
6.13	Primary stage of the best solution (from 20 runs) for Set-AT	130
6.14	Stage S_i of the best solution (from 20 runs) for Set-AT	131
6.15	Stage S_{ii} of the best solution (from 20 runs) for Set-AT	132
6.16	Stage S_{iii} of the best solution (from 20 runs) for Set-AT	133
6.17	Stage S_{iv} of the best solution (from 20 runs) for Set-AT	134
6.18	Sample output from the Set-AT solution	135
6.19	FROC curve for the set-AT problem	140
6.20	Graph of effect of feature set size	142
6.21	FROC curves showing fixed and co-evolving features	143
7.1	An example of preliminary work on retinal haemorrhage detection.	147

List of Tables

3.1	Pixel statistic features used in previous work	45
4.1	Parameters that make up a single zone genotype	52
4.2	The four zone types used in the co-evolutionary system	53
4.3	The effect of mutation on each parameter	55
5.1	GP parameters used for the full-image experiments	76
5.2	Full-image method result summary	97
5.3	Comparative analysis of fixed vs. co-evolutionary features	102
6.1	Parameters used on the multi-stage experiments	116
6.2	Summary of multi-stage method results	137
6.3	Solution size summary	138

Chapter 1

Introduction

The adaptability of human problem solving is one of the keys to its success. When presented with a new problem we can extract or derive pieces of useful information, *features*, from the environment whilst simultaneously formulating a plan to use those features to solve the problem. These two stages co-operate and complement each other, compensating for the other's weaknesses and exploiting each other's strengths. The division into extracting useful features and then learning to use them, is a useful metaphor for many learning systems and is heavily utilised in the fields of artificial intelligence (AI) and pattern recognition.

1.1 Examples

The ability to cope with new situations and escape hard-coded behaviours is one of the key aims in AI research. However, many AI methods still require humans to perform some key parts of the process. Frequently, human designers will perform the feature extraction phase, collecting and constructing features from the raw data that *they* think might be useful for solving the task. Then, a learning system, such as a decision tree or neural network for example, is trained to solve the task using those features. As an example, consider the problem of automated diagnosis of skin lesions. The potential benefits of early diagnosis of skin cancers has motivated a large amount of work into using AI techniques for automated diagnosis. In these systems, the human designer first extracts some key features about the lesion, such as size, symmetry, colour, fractal dimension etc. from the image.

These values are then fed into the learning system which tries to learn if the lesion is malignant or not. These techniques achieve reasonable performance, but still require a human expert to speculate about what might be useful features for the learning algorithm to use.

Consider a different example, in which items on a moving conveyer belt have to be identified in order to sort them – a very common industrial task. A human expert will decide on a number of features to be extracted from each object that describe it such as size, shape, weight, colour etc. These features are then passed on to a classifier of some sort in order to discriminate between the classes of object. Again, this is a simple example of a two-stage classification process in which AI techniques can achieve good performance.

1.2 Motivation

Both of the examples given in the previous section require a human designer to pre-specify the features that will be used for the classification. The most important consequence of this is that every time a new problem is attempted, a new feature set must be defined by hand. This not only introduces a bottleneck into the design process but also means that the learning process will forever be biased by the decisions made by the designer. There is no reason to assume that the designer picked the optimal set of features – in fact there is good reason to assume that they picked a severely sub-optimal set. As humans, we tend to choose features that are meaningful to humans, even though this meaning has no significance to the classifier. We also tend to choose different features to those that we would use ourselves, instead choosing features that are easy to extract and quantify. For example, in the above examples, texture may be a much more discriminatory feature, and also one that humans may use, but is difficult to measure in a meaningful way, so is not often used.

In the previous examples, the images would be acquired in relatively controlled environments, free from noise and cluttered backgrounds. This makes it relatively easy to extract problem specific features. In more complex real world applications, such as the car detection task shown in Figure 1.1, the opposite is true as human designers find it difficult to extract problem specific features from the image. In these situations, the approach taken is to choose highly generic feature sets which lie at a level much closer to the raw



Figure 1.1: A car detection task. (a) Shows the original image (b) Shows the ground truth. This problem has high intra-class variability, a very cluttered background, and difficult special cases (such as the car on the left which is almost entirely invisible due to a shadow). A human designer would find it very difficult to propose a good feature set for this sort of dataset. Image ©Selkirk Remote Sensing Ltd. Reproduced with permission.

data. For example, statistics such as brightness or pixel variance are used instead of more abstract notions such as length and width. Because of their generality, and because they cannot adapt, these feature sets have trouble capturing important aspects of the problem domain. The ideal feature set would be one that is generic, but also adaptable, so that it can harmonise itself with the data and with the classifier being used. This would remove the need for a human designer to specify features, but would allow the creation of domain specific features that are well suited to the classifier being used.

The dual action of extracting information and deciding how to best use it is one of the fundamental actions in a natural learning system, but one which is especially hard (and therefore often ignored) to replicate in artificial learning systems. This thesis focuses on the how we can begin to achieve this sort of behaviour in artificial systems, using simple visual *object detection problems* as a test bed. As its name implies, the goal of an object detection system is to find all instances of a specified type of object in a given image. These object detection tasks are an important sub-problem in the larger field of computer vision, and have many diverse applications.

The central theme running through this thesis is the division between the system that works out which visual features are useful and the system that decides how to use those chosen features. In previous work in the field the division is clear, and either one of the stages is fixed and inflexible so that the other can learn successfully. This work tackles this problem by removing the division, using techniques that can solve both parts simultaneously.

To find techniques capable of doing this, we look to nature for inspiration. Nature has an uncanny ability to create simple and elegant solutions to the most complex of problems, and uses evolution as a means to do so. The field of Evolutionary Computation (EC) captures some of the simpler aspects of natural evolution for use in problem solving, and has met with great success in many problems. The evolutionary algorithms used in this field often rely on very simplistic models of a single “species” which limits their power. Nature, however, contains billions of interdependent species which all exert evolutionary pressure on each other in what is called *co-evolution*. The interaction between co-evolving species allows much greater complexity to arise from much simpler components, which although not explicitly communicating, are all working towards a common goal. It is for this reason that the work in this thesis attempts to use co-evolution as a tool for the creation of a system that can simultaneously evolve visual feature detectors and methods for using them.

1.3 Thesis Goals

The primary goal of this thesis is to investigate the feasibility of simultaneously co-evolving visual feature extractors and algorithms that use those features. The very fact that this is possible at all would be an interesting contribution in itself as the idea of simultaneously learning two processes with such an integral co-dependence is contrary to what is normally considered possible. A system that can learn while its inputs are changing, not just in terms of value but in their very nature, at first seems like a strange and unproductive way in which to learn. However, the potential rewards are enormous – as the two stages are learning simultaneously rather than sequentially, they can complement each other and have the potential to be more concise and efficient than they otherwise would be. Also, if a system could learn at all in this incredibly noisy situation, the solutions it produced

would doubtless be more robust and tolerant than those produced by more traditional methodologies. This major goal contains many smaller sub-problems. In summary, this thesis aims to answer the following questions –

- *Is it possible to simultaneously co-evolve feature extractors and detectors?*
- *What methods are necessary for this co-evolution to be achieved?* This is non-trivial and involves issues such as overall architecture and underlying mechanisms of the co-evolutionary framework, as well as the representation and evaluation of feature extractors.
- *How effective would the co-evolutionary system be?* This involves testing the approach in different scenarios, both contrived and real-world, and analysing its successes and limitations.
- *How does this approach compare to traditional approaches?* This involves a comparison of both the solutions' accuracy and their compactness and efficiency.
- *What are the future implications of this approach?* If these co-evolutionary methods are successful at object detection, it could be possible to apply them to other two stage learning problems.

1.4 Thesis Structure

The structure of this thesis is as follows. Chapter 2 will explain the background knowledge necessary to understand the work and appreciate its scope and contribution. This includes an overview of the fields of computer vision and evolutionary computation, co-evolution, feature optimisation, and the related work in the field, with particular emphasis on object detection problems and pixel-statistic based classifiers which make use of *features* and *detectors*.

Chapter 3 illustrates some of the gaps in the field, examines a selection of work which tries to fill some of those gaps, and proposes using the co-evolution of features and detectors as a way to solve object detection problems effectively using no domain knowledge.

Chapter 4 describes the architecture and implementation of a framework in which features and detectors can be co-evolved together. This includes details of the co-evolutionary methodology, collaboration and credit assignment issues, feature representation and output calculation, and implementation details.

Chapter 5 describes experiments using the framework to coevolve solutions to object detection problems. The approach used is called the full-image approach due to the application of the solutions to all pixels in the image. Experiments on synthetic and real-world images are shown, and comparisons to fixed feature set methods are performed.

Chapter 6 describes a second set of experiments using the co-evolutionary framework. In these experiments however, the framework is used on top of a flexible multi-stage method, which allows the problem to be broken down into easier sub-problems and also uses significantly less computation. As with Chapter 5, an analysis of the solutions and comparisons to fixed feature set methods are performed.

Chapter 7 summarises the conclusions and contributions of the thesis and discusses some of the questions and issues that arise from the work, as well as discussing possible future developments and applications.

1.5 Publications Arising from This Work

- Mark Roberts and Ela Claridge. A Multi-stage Approach to Co-operatively Co-evolving Image Feature Construction and Object Detection in Franz R. Rothlauf et. al. eds., *Applications of Evolutionary Computation, Proceedings of Evolutionary Computing in Image and Signal Analysis (EvoIASP)*, Lausanne Switzerland, 30th March - 1st April 2005, LNCS Volume 3449, pages 396-406, Springer-Verlag
- Mark Roberts and Ela Claridge. Co-operative Co-evolution of Image Feature Construction and Object Detection in Xin Yao et. al. eds., *Parallel Problem Solving from Nature - PPSN VIII*, Birmingham, September 2004, LNCS volume 3242, pages 899-908, Springer-Verlag.
- Mark Roberts and Ela Claridge. An Artificially Evolved Vision System For Segmenting Skin Lesion Images. In Randy E. Ellis and Terry M. Peters, eds., *Proceedings of the 6th International Conference on Medical Image Computing and Computer-*

Assisted Intervention (MICCAI), volume 2878 of LNCS, pages 655-662, Montreal, Canada, 15-19 2003. Springer-Verlag.

- Mark Roberts, The Effectiveness of Cost Based Subtree Caching Mechanisms in Typed Genetic Programming for Image Segmentation in Gunther R. Raidl et. al. eds., *Applications of Evolutionary Computation, Proceedings of Evolutionary Computing in Image and Signal Analysis (EvoIASP)*, Colchester UK, 14th-16th April 2003, LNCS volume 2611, pages 448-458, Springer-Verlag.

Chapter 2

Background

This chapter provides the necessary background knowledge to understand and appreciate the contribution of the rest of the thesis. It starts by providing an overview of the fields of image analysis and evolutionary computation, and then illustrates some key concepts from the literature that will be built on in the rest of the work.

2.1 Image Analysis

The phrase “A picture is worth a thousand words” is one of the biggest understatements ever made – in most pictures, millions of words would be needed to convey the same amount of information as that gained by a human perceiving it, if it were possible at all. Even a blank image evokes perceptual, emotional and associative responses in humans that would require a huge amount of description. In many problems, the analysis of images can provide some of the richest and most useful information available and help to produce elegant solutions to difficult tasks. Ironically, however, even in the simplest images, there is such a large amount of useful information that finding and extracting all of it is an extremely difficult task.

The field of computer vision¹ aims to tackle this problem and extract useful pieces of information from image data. Generic computer vision systems that could perceive

¹The terms Image Analysis and Computer Vision are often used interchangeably, although strictly Computer Vision is concerned with understanding and perceiving the whole visual scene, whereas Image Analysis is concerned with extracting smaller pieces of useful information. The term Machine Vision is normally used to describe industrial inspection applications of these techniques.

and understand any given image are many years away, and currently the field focuses on smaller subtasks, normally for specific applications. Common subtasks in computer vision are segmentation, registration, object detection (localisation), object recognition, model fitting, tracking, and 3D perception.

The goal of segmentation is to separate an image into *meaningful* regions, which normally equates to finding the outlines of objects or areas in the image. Often, this is not possible and images are segmented on the basis of properties, such as colour or texture, that do not map directly onto objects in the real world. Image segmentation is the first step in techniques which rely on analysing the shape of a real world object. For example, the shape of a skin lesion is a criterion for assessing the lesion's risk of being cancerous, so a screening system would first segment the lesion in order to analyse its outline. There are many different techniques for segmentation, ranging from simple thresholds, to more advanced region splitting/growing, graph cuts and watershed methods.

In registration problems, the task is to align two or more images of the same thing, such that they can be accurately compared. Registration is very common in medical image analysis, where doctors often want to detect and measure changes in clinical features in medical imagery (such as X-Rays, CT, MRI, PET, SPECT, ultrasound, or normal photographs) taken at different times. If the doctor needs to measure if something has changed in size then an accurate registration is vital. Registration can of course be applied in three dimensions, as is often the case with volumetric imagery such as CT and MRI. The registration process is normally divided into two stages. The first of these is to find a set of landmarks present in both images, which could include points, lines, curves and surfaces. The second stage then tries to find a transformation between the sets of landmarks which accounts for the changes. Many different methods are available for this transformation generally they fall into three categories - rigid, affine, or non-rigid (allowing flexible “warping” of the image). Registration is also used to facilitate the “fusion” and visualisation of data from different imaging modalities i.e. an X-ray and an MRI scan of the same area could be fused into a single more detailed image.

Finding landmarks for registration purposes is an example of an object detection problem. In these types of problem, the task is to find the positions (and not normally the boundary as in segmentation) of a specific type of object in the image. For example an

object detection system might have to find all of the faces in an image of a crowd, or all of the cars in an aerial photograph. This is generally considered to be different to object *recognition* tasks, where a system has to assign a class label to an object which has already been found. For example, where an object detection system might find road signs, an object recognition would decide *which* sign it is. Generally, object recognition is regarded as an easier problem than object detection, as the system already knows that the image contains an object, allowing more assumptions to be made about the content. Object detection systems need to have the ability to search an image for any number of objects that may or may not be present. Dealing with the possibility of no objects being present makes the task considerably more difficult.

Object detection tasks will form the basis of the experimental work in this thesis. There are many ways that these problems can be tackled, but they normally involve learning for the reasons set out in Section 1.2. A wide variety of techniques are used, from more traditional pattern recognition techniques such as cross-correlation, principal component analysis (PCA), linear discriminant analysis (LDA), to more typical AI techniques such as neural networks, decision trees and evolutionary computation. Current favourites in the field are Lowe's SIFT algorithm [69] and Viola and Jones' boosted classifier cascade [118].

2.2 Evolutionary Computation

The field of Evolutionary Computation (EC) encompasses a wide variety of search and optimisation techniques which are inspired by the principles of natural biological evolution. As with much of artificial or computational intelligence, evolutionary computation is concerned with search. The search space and the methods of moving around it may be different, but the fundamental principles are still the same. In an evolutionary algorithm (EA), a *population* of individuals is used, each representing a possible solution to the problem at hand. Over a number of generations these possible solutions are selected and adapted using operators which are also based on ideas taken from natural evolution.

Whereas a standard search technique uses local changes to a single entity to search the space, the population based strategy used by evolutionary algorithms allows them to simultaneously perform many searches at different points in the search space. These ideas

have a number of very attractive properties such as being able to search a larger space, and the ability to more easily escape local minima.

There are several important common aspects to any evolutionary algorithm. The first of these is *representation* used to define possible solutions. Representation defines the nature of the search space and the possible operators we can use to move around it. There are different types of evolutionary algorithm for use with different representations. For example a *Genetic Algorithm* (GA) is commonly used for searching in problems where the representation is a fixed length string made up of a finite alphabet of characters (typically a binary string of 1s and 0s is used). *Genetic Programming* (GP) is an EA in which the solutions are actually fragments of computer programs, which, when run, are themselves the candidate solutions. If the solution can be represented as a fixed length vector of real numbers, the techniques such as Evolutionary Strategies (ES) or Evolutionary Programming (EP) can be used.

In each generation of an evolutionary algorithm, several key processes take place. All of the individuals are evaluated against an objective function and are assigned a *fitness* value depending on how well they perform. In a process based on the principle of “survival of the fittest”, this fitness value is then used to determine which of the individuals undergo operations that put them, or their offspring, into the next generation.

In natural evolution, there are generally two ways in which change can occur in the population. The first of these is mutation – a small change to the DNA which may or may not be useful in terms of the actual fitness of the individual. The second of these is by the breeding of two individuals. The breeding of two fit parents will hopefully produce fit offspring which have new qualities derived from both parents.

In an EA we have analogous methods for producing *artificial* offspring. A *mutation* operator is used to make small changes to an individual, and a recombination operator (often called *crossover*) is used to combine the properties of two separate solutions. The exact nature of these operators depends on the representation being used.

One of the most important issues in an evolutionary algorithm is the selection method used. A basic principle of Darwinian evolution is that fitter individuals are more likely to produce offspring than less fit individuals – giving rise to the notion of survival of the fittest. The *selection* of individuals for breeding is therefore proportional to the fitness of

the individual, but still inherently probabilistic. There are various different mechanisms for simulating this idea of selection in evolutionary algorithms.

In *Fitness Proportionate Selection* (FPS) the probability of selecting an individual is equal to its actual fitness divided by the summed fitness of all individuals. This type of selection is often referred to as “roulette wheel selection” as we can think about it as giving each individual a segment of the wheel with a size proportional to its fitness. We can then spin the wheel and select the individual that it stops on, and this will produce fitness proportionate behaviour. Fitness proportionate selection has a number of disadvantages however. It is very greedy, and favours exploitation rather than exploration, often resulting in premature convergence or stagnation of the population [32].

Tournament selection provides a fairer scheme than FPS. In tournament selection, we consider only a small proportion of the population at a time. A small set (normally between 2 and 10) of individuals is randomly chosen, and the fittest individual in that set is chosen. Tournament selection does not suffer from premature convergence or stagnation like FPS, especially with smaller tournament sizes as less fit individuals are more likely to be selected. It also has some other useful properties such as only needing relative fitness, rather than absolute values. Also, from an efficiency point of view, individuals only need to be evaluated if they are chosen for a tournament, which can sometimes produce a decrease in training time. Tournament selection is one of the most biologically plausible selection schemes.

In *rank selection* the individuals in the population are sorted by their fitness, and are then assigned a fitness based on some predefined distribution, such as linear or exponential. Rank selection does not usually permit domination by very fit individuals, as only their ordering is important, and thus premature convergence and stagnation are less of a problem.

2.2.1 Genetic Programming

Genetic Programming (GP) is an evolutionary algorithm in which each individual is actually a computer program. Essentially, GP allows us to create a program to perform a specific task. Genetic programming was popularised in 1992 by Koza’s detailed analysis

of its potential [55], but searches have revealed that GP was in use as early as 1987 by Schmidhuber [21], 1985 by Cramer [19] and maybe even as early as 1980 [111].

Currently, GP applications do not evolve what most people would consider to be programs, but something much more limited. A typical GP program is more like a single expression, and most could be written as a single line (although a long and unwieldy one) of code in most languages. They very rarely use any sort of modularity or iteration (although there are mechanisms to do this [55, 56]). However, even though the field is a long way from true automatic programming, the ability to automatically create programmatic expressions that perform specific tasks is extremely powerful and valuable. At the time of writing, there are “*36 instances where genetic programming has automatically produced a result that is competitive with human performance*” [61], according to the eight criteria set out in [57], which are as follows –

1. The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.
2. The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.
3. The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.
4. The result is publishable in its own right as a new scientific result $\frac{3}{4}$ independent of the fact that the result was mechanically created.
5. The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
6. The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.
7. The result solves a problem of indisputable difficulty in its field.

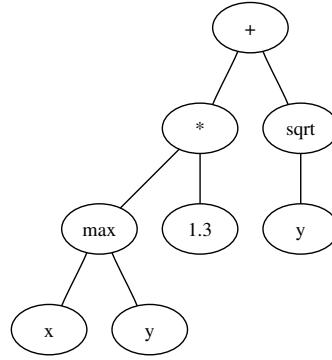


Figure 2.1: A example of a classical GP tree. This tree shows the expression $1.3 * \max(x, y) + \sqrt{y}$.

8. The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

2.2.1.1 Representation

As with any evolutionary algorithm, the representation used defines the possible solutions that can be generated i.e. the nature of the search space. In GP, we need to represent executable fragments of program code in a way that we can easily modify and evaluate them. The most commonly used variant is to represent programs as tree structures, although other (potentially more useful) representations are becoming more popular.

A sample tree representation is shown in Figure 2.1. The nodes of the tree are made from functions drawn from a *function set* which operate on leaf nodes taken from a *terminal set*. These sets are subject to a constraint called *closure* which states that they must be able to take the output of any other node as an input and produce a valid result. This normally also implies that the same datatype is used throughout the tree, although there has been work on typed GP which uses many different datatypes [73]. The use of function and terminal sets and the property of closure make it very easy to randomly generate trees of a specific shape and size, which is needed to create the initial population, and to generate subtrees for the mutation operator described in the next section.

Tree based GP often suffers from a problem called *bloat*, in which the average solution size in the population increases exponentially, without a corresponding rise in performance.

This growth has a proportionate effect on the evaluation time of the trees. Many different methods have been proposed for limiting bloat, but it still remains a significant issue in most systems.

Other Representations Tree based representations were the logical choice when GP was first conceived, due to their parallels with recursive representations used in languages such as LISP and Prolog. Now, however, as we attempt to solve ever more complex problems, the limitations and expressiveness of tree based representations are becoming clear. One of the next logical steps is to use a graph based representation, which shares many of the same benefits as trees, but also provides a more expressive syntax and the easy reuse of subcomponents of the program.

There have been many different attempts in the literature to evolve graph structures including the Parallel Distributed Genetic Programming (PDGP) system [87, 88, 86], the TAG architecture [80, 79], Cartesian Genetic Programming (CGP) [72], linear graph GP [49], Parallel Algorithm Discovery and Orchestration (PADO) [116].

At the other end of the spectrum to graph based GP is *Linear GP* which works with a completely flattened representation of just a single sequence of operations. These operations are normally simple instructions which perform arithmetic operations on a number of predefined variables. Often, the operations are very similar to those used in assembly languages. In fact, in some systems machine level instructions are actually used thus eliminating the need for parsing, resulting in a speedup of several orders of magnitude [76].

2.2.1.2 Genetic Operators

The tree-like structure of classical GP was chosen partly due to the ease by which it can be adapted. Assuming that the closure property described in Section 2.2.1.1 holds for all nodes in a tree, we can freely add, remove, and modify subtrees at any point without any possibility of generating invalid trees.

The crossover operator in GP is usually implemented by swapping randomly selected subtrees² from two parent trees. This process is illustrated by the example shown in Figure 2.2. Once the crossover points are chosen, the subtrees below those points are removed and inserted at the crossover point in the other parent. This type of crossover ensures that both children have genotypic properties of both parents.

Mutation is a similar process to crossover, but uses only one parent. An example of a mutation operation is illustrated in Figure 2.3. A random point in the tree is selected and the subtree beneath that point is removed. A new randomly generated subtree is inserted at the mutation point. The depth of this new subtree is normally limited to a user defined value. One child is produced from the mutation operation.

There are of course alternatives to these simple operators which offer different advantages such as helping to reduce bloat or preserving the context of the subtrees [51, 64], however their basic aim remains the same – to produce children which have some properties of the parents, and to maintain diversity in the population.

2.2.1.3 Evaluation

Due to the tree based representation used, the evaluation of a GP tree is relatively straightforward. A simple depth first recursive traversal of the tree will eventually produce a single result from the root node. Depending on the underlying implementation, the actual evaluation procedure might be quite different (see [52] for a discussion of implementation) but the basic principle is always the same.

The act of traversing the tree to evaluate the result of a GP tree is a notorious bottleneck. We normally think of programs as executing very quickly, especially when they are quite trivial as typical GP expressions are, but we must remember that these programs are not compiled. The GP programs are parsed at runtime, so the act of traversing the tree and storing the result is many times slower than a compiled version of the tree. If we then consider that GP is a population based system, and that each tree will be evaluated

²In most systems the subtrees are not truly randomly selected. As trees grow, the number of leaf nodes massively outnumbers the number of function nodes. In this situation a uniform random choice would normally result in the selection of leaf nodes, and so crossover would simply be making inconsequential swaps of terminals. The selection of subtrees is therefore weighted to ensure that most selections (90% in most cases) are function nodes.

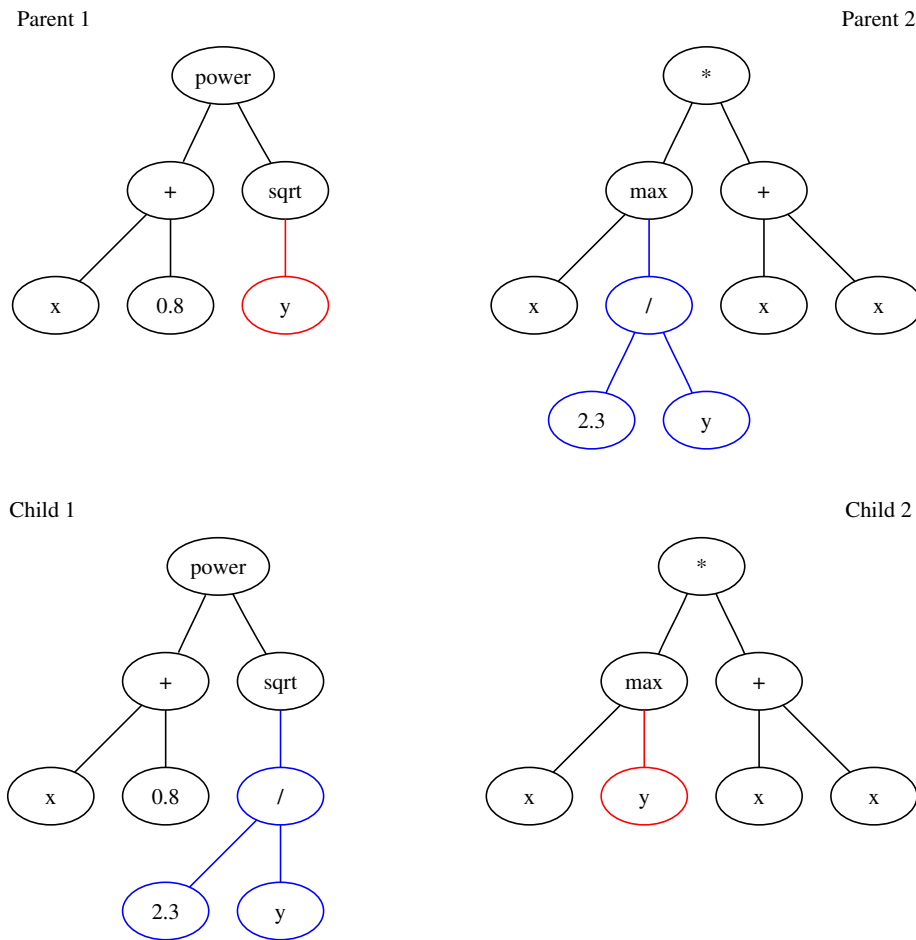


Figure 2.2: Demonstration of the simple crossover operator for GP trees. In each parent, a subtree is selected (shown in red and blue). These subtrees are then swapped to create two new children.

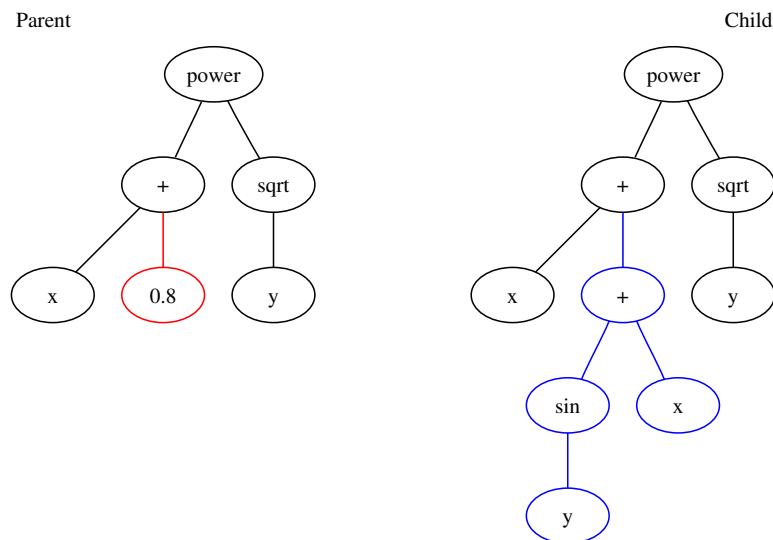


Figure 2.3: Demonstration of the simple mutation operator for GP trees. In the parent, a subtree is selected (shown in red). This subtree is deleted and a new, randomly generated, subtree is inserted in its place (shown in blue).

many times per generation, it becomes clear that any small bottleneck will be multiplied millions of times and become a major problem. This bottleneck is probably the single biggest disadvantage and is often blamed for GP’s lack of use in some application areas.

Understandably, there have been many attempts to speed up the evaluation of GP trees. A particularly interesting alternative is to use *vectorised evaluation* [50] which reduces the tree parsing overhead by evaluating all test cases at each node in the recursion, meaning that the tree is only parsed once. The speed of evaluation can be quite significantly increased by using caching mechanisms. Due to the nature of the evolutionary process, repeated subtrees are very often found in large numbers in the population [65]. It therefore makes very little sense to re-evaluate them each time they are found. Various systems have been proposed to make the process of caching results possible [33, 29, 50], often resulting in very significant speedups. One disadvantage of these systems is that they can only be used when evaluations have no side effects on any global state. In previous work, we have shown how a heuristic measure of operator cost can be used to improve caching systems by only caching trees where the cost of evaluation is significantly more than the cost of caching, search, and retrieval [101].

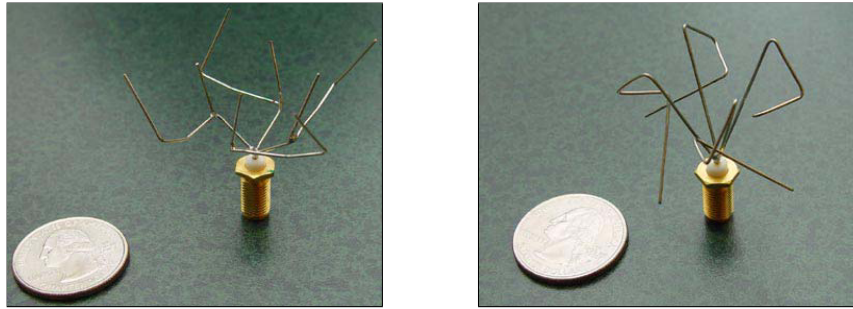


Figure 2.4: Antenna designs evolved in [68] and [67]

2.2.1.4 Applications of Genetic Programming

Genetic programming is clearly a powerful method for automatically designing programs, but more recently has seen many uses in physical design processes. Electronic circuit design is a prime example as the mapping from programmatic structures to circuits is trivial. Designs for both digital [55, 56] and analog circuits [57, 61, 60, 59, 58] have been successfully created, with human competitive results in many cases.

It is also possible to use genetic programming for designing physical entities. The most famous example of recent times is the evolution of the antenna used on NASA's *ST5* spacecraft. The antenna design needed to meet a very challenging set of constraints laid down by the mission designers. GP was used [68, 67] to design an antenna, using programming constructs similar to those used in turtle graphics, to effectively *draw* the shape of the antenna. The final designs (see Figure 2.4) met all of the constraints and used a novel design of antenna never seen before.

2.2.2 Co-evolution

In general, co-evolution refers to the evolution of several different species, where the species are dependent on each other in some way to the point of exerting a selective pressure on each other. The typical example of co-evolving species normally given is that of pollination. Species such as bees or hummingbirds often co-evolve with the flowers in their ecosystem. The bees need the pollen and nectar that the flowers can provide, and the flowers need the bees to spread the pollen in order to reproduce. This dependence leads to changes in the biological properties of each organism – the flowers can adapt their shape and colours to

be more visible to the bees, and the bee’s senses can adapt to be able to find the flowers more easily.

In a successful co-evolving system, all of the species get what they want, and often with less effort than if they were not co-evolving. We can see a co-evolving system almost as a modular “divide and conquer” approach to problem solving. The different species are each evolving part of the solution to a problem (whether they know about the global problem or not), and the explicit or implicit co-operation between them enables them to pool their resources.

This increased problem solving power, implicit co-operation, and often simpler solutions, made co-evolution attractive to the evolutionary computation field, where these properties are highly prized. There are two major types of co-evolutionary technique used in evolutionary computation – *competitive co-evolution* and *co-operative co-evolution*.

2.2.3 Competitive Co-evolution

In competitive co-evolution the different species have different goals and they increase their fitness at the expense of the other populations. Typical examples of competitive co-evolution occur in predator-prey situations where the two species are evolving the ability to hunt and avoid being hunted respectively. Competitive co-evolution gives rise to an “arms-race” situation where there is no overall goal, apart from staying ahead of your competitor (also known as a zero-sum situation). In evolutionary computation terms, this arms race has the side-effect of causing an increase in the fitness of both species, where none would occur in a single species environment.

A good example of the use of competitive co-evolution is the work of Hillis [39] in evolving sorting networks. A sorting network is made up of gates which test two numbers for the correct order and then swap them if they are incorrect. A number of these gates can be organised (some in parallel as long as the inputs are different) such that the output of the network is a sorted list of the input values. These networks should be as small as possible, so the problem of finding the smallest sorting network for a given number of inputs has been widely used as a test problem for decades. Hillis tackled the problem of sorting 16 integers. When evolving a 16 input sorting network, fitness can be calculated by testing all of the 2^{16} different permutations of the 16 inputs and counting those that

were sorted correctly and those that were not. This is a very large amount of computation and Hillis found that although testing in this way produced reasonable results, they were not competitive with hand-crafted solutions.

Hillis then introduced the concept of competitive co-evolution into his system. Instead of measuring fitness on all of the 2^{16} examples, he used a “parasite” that was constantly evolving a small set of hard test cases, which were presented to the sorting network. Therefore, one population was evolving hard test cases for the other population which was evolving the sorting networks. This formed an arms race where each population was constantly trying to stay ahead of the other, until eventually a sorting network of the specified size was found. As only a small sample of test cases was used, the process was significantly faster, and did not get trapped in local minima as often as the method using all of the test cases.

The artificial arms race induced by competitive co-evolution is a useful problem solving tool, and often allows less computation to be performed overall. It also more closely follows a classical learning process where a teacher continually challenges a student by coming up with harder and harder exercises. This mechanism allows the learning process to proceed along a more natural path, rather than trying to learn about absolutely everything at the same time.

2.2.4 Co-operative Co-evolution

In co-operative co-evolution [92], each species forms a part of entire solution to a global problem and each is dependent on the performance of the other species in some way. The species involved may not know anything about the global problem, but become a part of it due to the environment in which they live. This is the type of co-evolution present in the bees and flowers scenario. Each species is trying to solve their own particular problems, but the interaction with other species implicitly exerts a selective pressure on them and vice versa.

Introducing co-operating species into the evolutionary computation domain allows problems to effectively be decomposed into smaller sub problems. The implicit or explicit co-operation between the species reduces redundancy and promotes generality amongst the

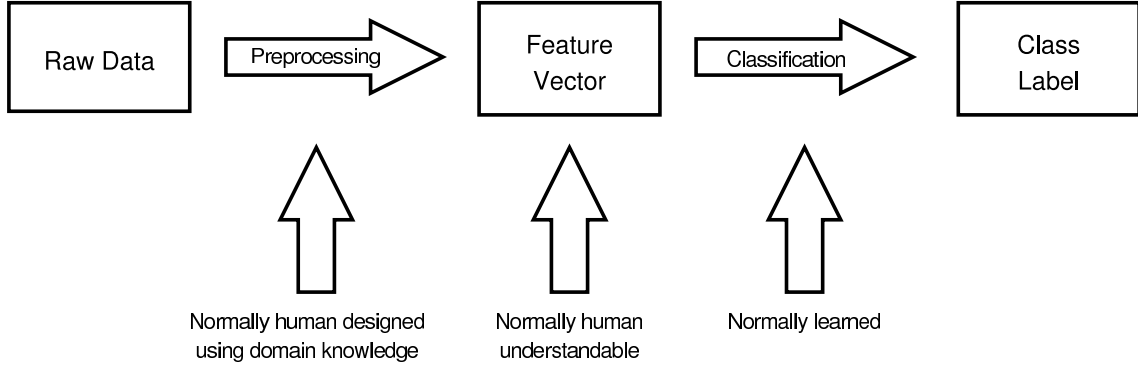


Figure 2.5: The traditional classification pipeline. The raw data is first preprocessed to extract possibly meaningful pieces of information. The classification stage then uses this information to learn a solution to the problem.

components. These are desirable goals in all areas of problem solving and are especially useful in population based search techniques where computation is expensive.

Co-operative co-evolution is a relatively young field and has only begun to be developed since standard methods and architectures were defined in the last few years [92, 93]. However, the co-adaptation of subcomponents can be seen in earlier incarnations such as classifier systems, in which a set of rules are evolved to form a collective behaviour.

2.3 Classification Pipeline

In all areas of problem solving, be they in the real world or in the realms of artificial intelligence, there is very often a common process that is used. Given the raw data of the problem domain, the first step is usually to pre-process that data, and pick out key pieces of information, known as *features*, which may be useful in solving the problem. These features are then used by a second stage, often called detection or classification, to actually produce a solution to the problem. This two stage classification pipeline is the basis of many machine learning techniques as it is a very intuitive process which mimics the way in which humans think about problem solving. However, the first pre-processing stage is often given very little attention and the learning techniques are applied only to the second classification stage. The classical pipeline can be seen in Figure 2.5.

2.4 Feature Optimisation

In the first, pre-processing stage of the classification pipeline, we try and reduce the dimensionality of the data by extracting (or creating) some relevant features from it. The exact nature of the features is governed by the task, and the problem solving method being used.

The term “feature” is only very loosely defined as a piece of information that may be useful for solving a problem. This all-encompassing definition sometimes leads to a very confusing and inconsistent array of uses of the term. In the literature, the term is used to mean anything from a raw numerical pixel value, an edge or a line, an abstract object such as a triangle or a corner, right up to an actual real-world object such as a car.

The task of extracting/selecting/creating appropriate features for any given task is a remarkably hard task, which depends on the problem, the format of the data, and the method being used to solve it. In spite of the difficulty of the task, we find (as in all areas of AI) that humans appear to find it relatively trivial. When asked to solve a problem, we can very quickly decide how we would solve it whilst simultaneously deciding how to extract the information we would need to solve it.

Given a set of data to learn from, it is reasonable to assume that its default form is probably not in the most suitable form for solving the task. This is a reasonable assumption, as there are an infinite number³ of ways of filtering and re-representing the data. It would be an extraordinary coincidence if the default format of the raw data was the most useful form for learning a solution to the problem. For this reason, the field of feature optimisation emerged to find ways of transforming data into a more easily learnable form. There are three main techniques for accomplishing this – *feature subset selection*, *feature extraction*, and *feature construction*, which will be explained in the following sections. Due to the nature of feature optimisation techniques, evolutionary computation is a useful tool for solving these problems, and some approaches to evolutionary feature optimisation will be discussed.

³This is the case when feature construction with addition is used. See section 2.4.3

2.4.1 Feature Subset Selection

As its name implies, feature subset selection (FSS) is the process of selecting a subset of the features to use in the system. This has the obvious advantage of reducing the dimensionality of the problem, which is always desirable in problems like this due to the infamous *curse of dimensionality* [6]. With most problems we can reasonably assume that not all of the features present are useful for solving the problem, so FSS aims to find a smaller subset of those features that have equal or greater problem solving utility. More formally we can state that, given an initial set of features $F = \{A_1, A_2, \dots, A_n\}$, we would have after feature selection a smaller set of features $F' \subset F$.

Immediately it is clear that the problem of feature subset selection could be easily tackled by a classical binary genetic algorithm. If we have N features, then the bits of a binary GA with gene length N could be used to represent the features' inclusion in the subset. Applying a classification technique to this subset would provide fitness values for the gene. This is a widely used technique, first proposed by Siedlecki and Sklansky in 1989 [105].

2.4.2 Feature Extraction

Feature extraction is a slightly confusing name for a technique which does not strictly *extract* features. Feature extraction is basically a term for any technique that performs some functional mapping of the original feature set onto a lower dimensional space. This could be anything from a simple weighting of the features to something more complex like a principal component analysis. A more useful way to imagine the process is that it changes the shape of the search space as described here by Raymer [97]

The result is that the feature space is expanded in the dimensions associated with highly weighted features, and compressed in the dimensions associated with less highly weighted features. . . This allows the . . . classifier to distinguish more finely among patterns along the dimensions associated with highly-weighted features.

More formally, given an initial set of features $F = \{A_1, A_2, \dots, A_n\}$, feature extraction will produce a new feature set $F' = f(A_1, A_2, \dots, A_n)$.

Raymer [95] used GP to evolve feature modification functions, one per original feature, which transformed the original linear features into highly non linear functions, which allowed a k-nearest neighbour classifier to achieve a performance gain over the original datasets. In [96], the same author uses a similar technique, this time using a GA to optimise a matrix of weights for the feature extraction process. Also in this work, a binary GA is evolved, enabling a feature subset selection to be simultaneously performed on the extracted features.

2.4.3 Feature Construction

As its name implies, feature construction involves the creation of new features that aim to re-represent the problem space in a more useful form. New features are constructed from the original set, normally by creating composite features which combine some of the original features in some way. These features can either be added to the original feature set, which will result in a larger feature set, or are used *instead* of the original set. This second form, feature construction with replacement, has the ability to reduce the size of the feature set. Diagrammatic examples of feature construction can be seen in Figures 2.6c and 2.6d.

Genetic programming can be applied to the problem of creating more useful composite features from the original set. Smith and Bull [108, 109, 110] used this approach, to evolve sets of non-linear composite features, which were used in the training of decision trees (using the C4.5 method [94]), the output of which provided fitness values for the GP population.

Bhanu [8], used co-operative co-evolution to perform a similar task, creating composite features from the original dataset. A classifier was induced using the new feature sets and used as fitness. The improvement over standard GP was statistically significant on the classification problem used (the “glass” problem from the UCI machine learning repository [75]). The use of explicit co-operation here ensured that the elements of the feature set concentrated on separate parts of the problem and avoided redundancy.

The ZEUS system [25] uses genetic programming to evolve sets of feature constructors which reformat the raw data. The fitness of these features is given by the performance of a support vector machine using the features on a classification problem.

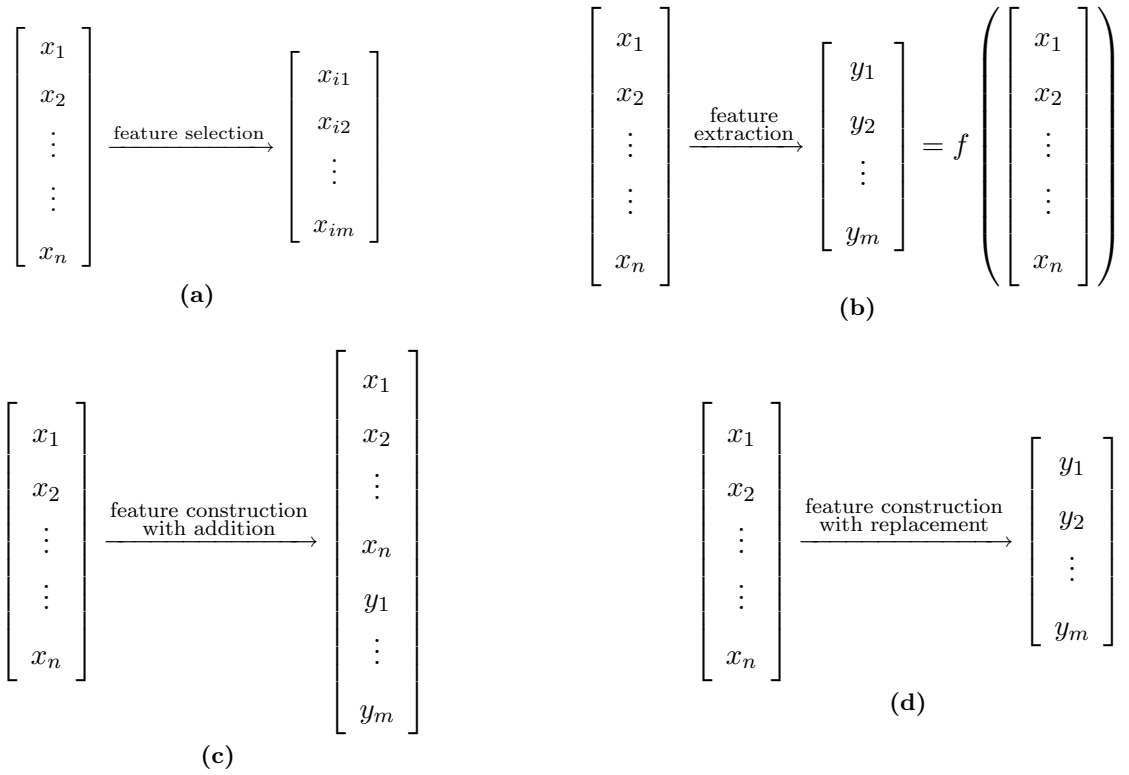


Figure 2.6: Diagrammatic representations of 4 feature set transformation methods (substantially modified from [104]).

2.5 Evolutionary Computation for Image Processing and Analysis

2.5.1 Overview

The magnitude and complexity of image processing tasks means that developing algorithms to work with images requires a great deal of skill and experience. It comes as no surprise therefore to see that the problem solving power of evolutionary computation is frequently applied in this area. The interest in EC solutions to imaging problems has always been present, but as both EC and image processing require large amounts of computation, the field only really took off in the late 1990s as the computing power needed became more accessible. In this section we will briefly review past work on EC approaches to image processing (normally regarded as transforming one image into another more useful form) and image analysis (extracting some meaningful symbolic information from an image). Section 2.5.4 will then focus on *Pixel Statistic Based Classifiers* as these will provide the basic ideas upon which the rest of the thesis is based.

2.5.1.1 Image Processing

Evolutionary Computation is well suited to the transformation-type operations used in image processing. In these sorts of tasks, we normally want to transform the image into a more useful form. The definition of what is “more useful” is dependent on the task. The best example of this would be image restoration. In restoration tasks we need to remove noise, alter contrast, sharpen, blur etc. to produce a cleaner, clearer version of the original. The order and parameterisation of these operations requires expert knowledge, and is a good candidate for evolutionary search. Evolutionary computation has been successfully applied to a number of generic restoration problems [74, 122, 40], specific restorations such as noise reduction [112] and specific tasks such as damaged document restoration [99, 98].

Other work in this category includes the evolution of pseudo-colouring algorithms. For example, [90] uses GP to enhance greyscale brain MRI images by colouring them. As the goal in this work was to create a perceptually clearer image, a human was used to assign fitness values. In [70] the aim is to colour evolved art to make it more interesting.

Image compression has long been an attractive proposition for GP. An image can be seen as being an intensity function which returns brightness at any given x, y point. If a program could be found that approximated that function, then it would be a programmatic representation of the image. This is simply a symbolic regression problem, so GP can be applied quite easily. If the programmatic representation of the image is smaller than the original data, then compression is achieved. The GP system can be forced to produce small solutions in order to achieve a specific compression ratio. This type of programmatic compression was first tried in [55] with toy images, and then used on real world images (and audio) in [77] although the quality of the compressed output was not very good. This quality problem was analysed and improved upon in our previous work [100] by using a fitness function based on frequency information derived from a discrete cosine transform. This produced better compression using significantly smaller programs. Other work on programmatic compression includes [11] and [62]. GP has been used to evolve predictors for lossless image compression in [30], wavelet compression schemes [53], while others have used GP to preprocess an image into a more “compressible” form [78].

2.5.2 Detection and Classification Systems

By far the largest group of works applying genetic programming to images could be broadly categorised as *detection* problems – that is a system that takes in an image and outputs some meaningful information about that image, either outputting another image, or some more abstract symbolic information. This broad definition covers everything from segmentation tasks, “standard” image feature (edges/corners/lines) detection tasks, through to more abstract detection tasks such as finding all of the dogs/cars/clouds/planets in an image.

There have been many approaches to using GP to evolve detection and classification systems. These systems vary greatly in many respects, with different input and output formats leading to many different functions and methods used. These distinctions are often used to classify these methods, but they are all in fact fundamentally very similar. This section will describe some of the previous work in this area and then Section 2.5.4 will describe one of these methods in more detail.

One of the first large scale uses of genetic programming for image analysis was by Poli in 1996 [85, 83]. This was one of the first works to use images as the datatype for the solutions – that is that all of the terminals were images and all of the functions processed and returned images. As discussed in Section 2.2.1.3, even using simple datatypes the evaluation of a GP is very slow. Using images as a GP datatype requires many times more computation and this was one of the first works which realistically attempted to use this type of method, and despite being hampered by the enormous amount of computation, still produced good quality results. This work used a terminal set comprised of four items – the input image itself and mean filtered versions of the image at three different sizes, 3x3, 7x7, 15x15. The functions used were the 4 standard arithmetic operators and the min and max operators. These functions were applied to each pixel of the inputs and produced a new image as output. The technique was applied to two medical imaging segmentation problems and in both cases produced visually superior results although by the author’s admission, the enormous computation time hampered the gathering of more statistically sound results. In spite of this, the work opened up the field and showed that GP could be used in this way. The author postulated that in the future GP would become a powerful tool for developing image analysis algorithms.

This type of approach using images as the GP datatype, which from here on will be called “image-based” approaches, will always be many orders of magnitude slower than GP based on primitive datatypes. For example, a function that adds two numbers together has to perform only a few instructions. An operator that adds two images together will perform a number of additions proportional to the size of the image. Currently this figure is in the order of the hundreds of thousands, but could conceivably be millions for a real world application. For this reason, using image-based approaches on anything other than toy problems will, for the foreseeable future, be limited to those who have vast computing resources and/or specialist imaging hardware.

Aoki and Nagao [4] present a system which is very similar to Poli’s, but uses a richer function set containing 19 different functions in total. These functions included traditional image processing operators such as edge detectors and thresholds as well as more mathematical operators such as sums and products. Their system was applied to the tasks of

removing handwriting (but leaving printed characters) from documents, and finding the boundaries of cells in histological images.

The GENIE system [117, 14, 113, 34, 81, 13, 114, 15, 35, 16, 36, 23, 82], is an image-based system which has been used for remote sensing applications on Earth and Mars [82]. GENIE, which stands for GENetic Imagery Exploitation, is based on a form of linear GP and is applied to terminals consisting of different spectral bands of IKONOS satellite imagery. Fixed length chromosomes are used and a set of “scratch-planes” are available for storing temporary results. This use of scratch-planes (akin to registers in classical linear GP) means that GENIE is actually evolving directed acyclic graphs. Each gene in a GENIE chromosome describes the operation performed, where the input is taken from, where the output is written to, and the parameters given to the operator. A large function set is used which contains mathematical and image processing operators, all of which take one or more input planes (either the from the terminal set or from a scratch plane) and produce one or more outputs which are written to the scratch planes. The user nominates one of the planes to be taken as the final output of a system, which is then used for fitness calculation. The GENIE system is very powerful and produces better results than traditional deterministic classification techniques on the same problems. Because of its linear structure, the solutions produced by GENIE are easily understandable and modularity is exploited. The system is applied to tasks such as the detection of forest fire scars, roads, water, clouds, beaches and golf courses! Even though the gene uses some numerical parameters, the GENIE system is a image-based approach with each operator inputting and outputting image planes. The processing speed is helped by the small fixed length chromosomes and a simple graph analysis step which allows certain genes to be ignored if their output is never used.

Marc Ebner used an image-based method to evolve an *interest operator* [26, 27]. The aim of this operator was to find useful landmarks for a motion tracking problem. His multi-objective fitness function was therefore based on the quantity and quality of useful points found and the performance of a motion tracker using them. His function set consists of a selection of binary and unary arithmetic and image processing functions.

Bhanu [9] showed extensive work on using a image-based system to detect various objects (people, cars, roads, lakes) in SAR and IR imagery. The 16 terminals used were

filtered versions of the image, containing mean, standard deviation, min, max and median values over 3 scales (3, 5 and 7 pixels). Seventeen operators were used which consisted of simple arithmetic and statistical functions and were applied to each pixel of the input images. The output images are thresholded to produce a binary image. Fitness was calculated by comparing the overlap of this binary image with the ground truth. The results shown were impressive and the solutions produced were surprisingly simple and easy to understand. However, this approach benefited greatly from not training on the whole image, but “only to a region or regions carefully selected from a training image”. Also, the image datasets used to test the method were not highly challenging, presenting little in the way of cluttered backgrounds etc., and in many cases reasonable solutions could have been achieved using simple thresholding operations.

2.5.3 Geometric Property Classifiers

The image-based methods described in the previous section provide an extremely powerful paradigm in which to evolve sequences of operations on images. However, their hunger for computing power makes them less than ideal in practical applications. For this reason, much of the early work in applying GP to image analysis problems is based on the idea of using classical image processing techniques to segment out an object from an image, and then using GP to evolve an arithmetic expression to classify this simpler representation.

In 1993, Tackett [115] used this approach in a target detection application, detecting military vehicles in infrared imagery. The first stage of the system uses a standard target recognition algorithm to very crudely extract any “blob” that could possibly be an object. Twenty geometric and moment-based properties of this blob are extracted such as size, shape, symmetry, average grey level etc. These properties become the GP’s terminal set, and using a function set of arithmetic operators, the system must evolve a classifier that can discriminate targets from non-targets. This type of approach is significantly faster to train than an image-based approach, as the first stage effectively samples from the image and only presents a subset to the evolutionary system.

In [48], Johnson uses a similar system. In this system, the task is to find the coordinates of the hands in a pre-segmented silhouette of a person. The extracted terminals include the centroid of the silhouette, and the top-left and bottom right coordinates of the bounding

box. The GP system used was multiply typed, allowing representations such as floating point numbers, coordinate pairs (points), and point lists. The functions used return data from the image such as edge locations, and other functions can manipulate these points and lists of points until a single point emerges from the root node of the tree. This coordinate is compared with the known hand position in order to calculate fitness.

2.5.4 Pixel Statistic Based Classifiers

The geometric property classifiers described in the previous section have the disadvantage of requiring a custom preprocessor to be created by a human designer. It would be better if a system required no human input and didn't need a preprocessor to supply it with data, but could instead find a solution using nothing more than the raw pixel data available in the image. However, as there is no preprocessor, all pixels in the image must be considered by the evolving system. This provides a very flexible way of evolving a system, but the computational requirements are the same as for the image-based approach considered earlier.

Pixel statistic based classifiers will form the basis of the work in this thesis, so this section details the steps involved in evolving a system like this for an object detection problem. In an object detection system, we are not interested in achieving a precise segmentation of a visual feature, but instead we merely want to establish its presence (or lack of it) and its position. In order to achieve this, a core set of actions is normally performed, which is common to most methods seen in the literature. These actions are conceptually similar to the stages in the classification pipeline shown in Figure 2.5, with only the last stages being specific to the problem domain of visual object detection.

Evolved object detection systems normally use an approach whereby a number of features are extracted at every point in the image. The values of these features are then used as terminals that an evolving population of GP trees may use. The output of the GP tree at each pixel is taken to be a classification of that pixel, and these outputs are then further analysed to calculate the fitness of a GP detector. These 4 steps, which we will call feature extraction, detector application, blob extraction and blob analysis, are described in detail in the following sections.

2.5.4.1 Feature Extraction

Conceptually, the feature extraction stage performs a very intuitive action – extracting relevant information about the image that can then be used to assist the classification stage. It is what humans do implicitly every time they look at something. If a person was asked to look at a picture and describe the weather it showed, they would instantly extract relevant features such as the colour of the sky, the brightness of the scene, the presence of shadows, what people were wearing, the presence of rain/water etc. The values of these features are then used as a basis on which to perform a classification i.e. “it is raining/sunny” etc.

In an object detection system, we don’t have access to such high level features. In fact, in a generic system we don’t really have any ideas about what the features are that would help us make a classification decision. Instead, lower level features are used which are normally simple statistical or geometrical properties of pixels or regions in the image. A predetermined set of these low level features is used, which hopefully captures enough detail about the problem domain to then be useful to the next classification stage. Also, unlike the human visual attention system, the system does not know where the features might be, so instead has to extract these features at every pixel by sliding the feature set across the image, and calculating the value at each point.

A “feature” in the case of these systems is always positioned relative to the current pixel of interest, although not necessarily centred around it. An example feature set used in the literature is shown in Figure 2.7. This is a feature set proposed by Zhang [124] and consists of 20 zones, half of which calculate the mean of the pixel values under them, and half of which calculate the standard deviation of the pixel values.

The feature extraction zones are swept over the image, and at every pixel, each feature zone is overlayed relative to that pixel, and the corresponding statistic is calculated about the pixels that it covers. This will produce a number (20 in this case) of “feature planes” which hold the value of that feature at every point. An example set of feature planes generated using this feature set is shown in Figure 2.8.

At the end of the feature extraction phase, we therefore have a number of feature planes, which although they are visualised as images here, are in effect only used as lookup tables.

	Mean	Std. Dev.	Feature shape
	F1	F2	Big square A1-B1-C1-D1
	F3	F4	Small square A2-B2-C2-D2
	F5	F6	Upper left square A1-E1-O-G1
	F7	F8	Upper right square E1-B1-H1-O
	F9	F10	Lower left square G1-O-F1-D1
	F11	F12	Lower right square O-H1-C1-F1
	F13	F14	Long horizontal axis G1-H1
	F15	F16	Long vertical axis E1-F1
	F17	F18	Short horizontal axis G2-H2
	F19	F20	Short vertical axis E2-F2

Figure 2.7: Feature set used in [124]. The table lists the features used which are defined by the grid.

For any pixel in the image, we can quickly look up the data in these planes and create a feature vector containing the values of the different features at that point.

2.5.4.2 Detector Application

The next stage is to actually apply an evolving GP tree to the data. To evaluate the fitness of a given GP tree, it is “run” at every pixel in the image. At each point, the values of the pre-extracted features are looked up and the terminal set is instantiated using their values. During the execution of the tree at that point, it will therefore have access to the corresponding feature values for that point, and can use them to produce its classification. For each pixel, the GP tree is evaluated and a numerical result is produced, as shown in Figure 2.9. This numerical output is the first indication of the classification, but its continuous range is not in itself useful for discrete classification. The next stage, blob extraction, is used to tackle this issue. It should be noted here that the output value of the detectors is normally unbounded and can take any positive or negative value that the datatype will allow.

2.5.4.3 Blob Extraction

The real valued map produced by the Detector Application stage is still not in a form that can be used to calculate fitness, and needs to be converted into a form that is useful for object detection problems. By this we mean that we require a representation in which we

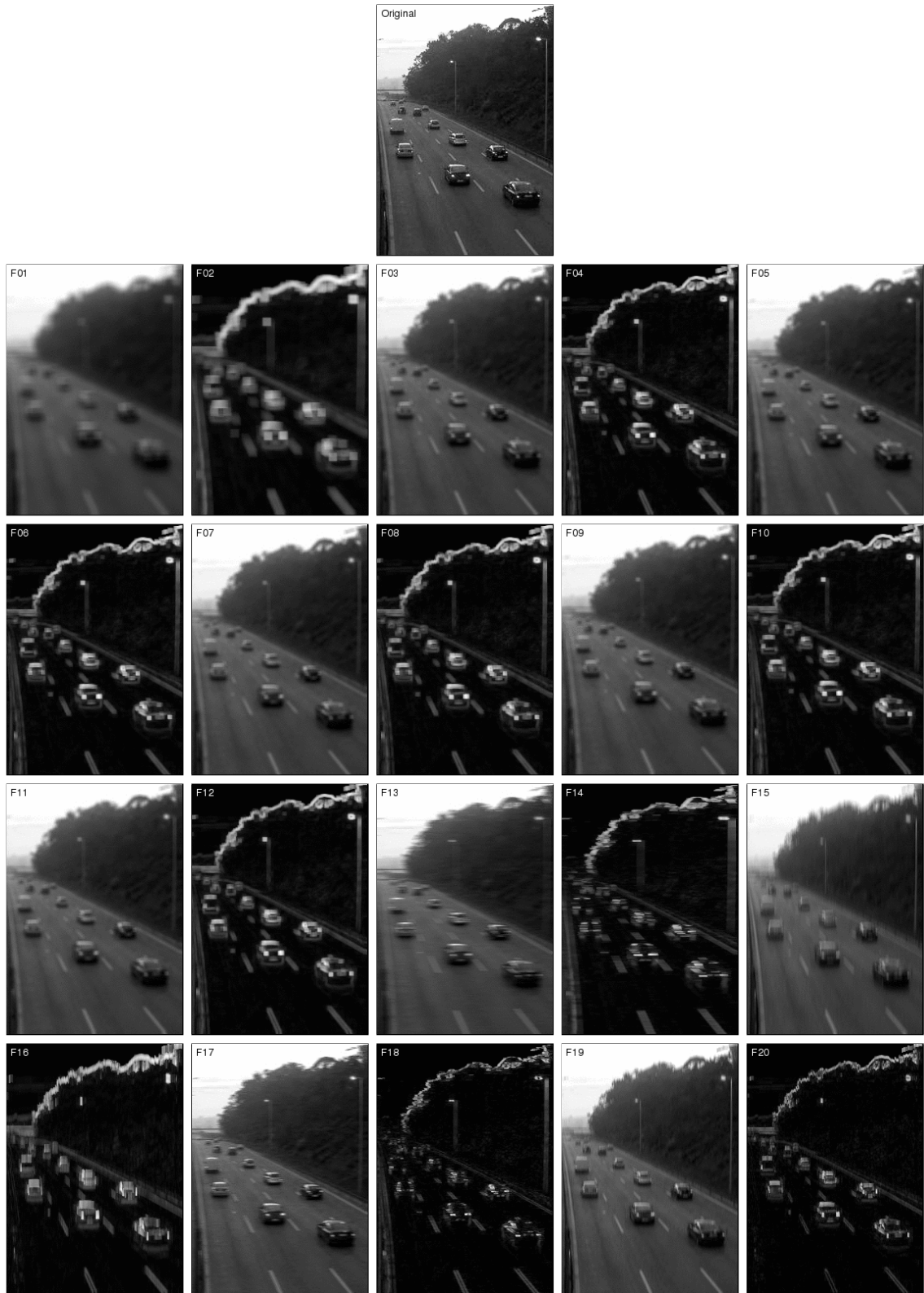


Figure 2.8: The top image is the original. The others are the feature planes that result from applying the 20 feature extractors shown in Figure 2.7. All images are normalised for correct display.

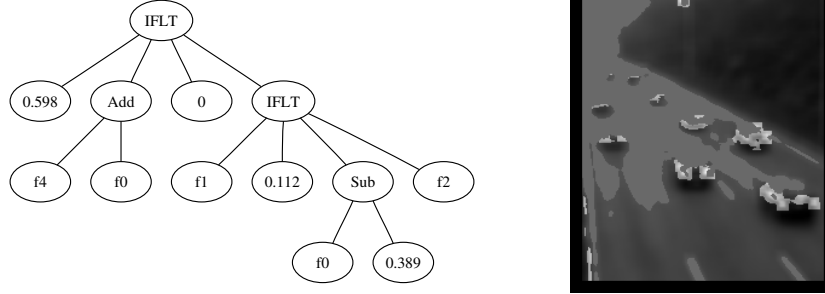


Figure 2.9: Shows an example detector tree and its output. The detector tree (left) is applied to the road image and produces the output shown (right, normalised for display purposes). The detector tree uses terminals, $\{f_0, \dots, f_{20}\}$ which refer to the pre-extracted features calculated in the previous stage.

can both determine the class of the pixel as being from a finite set of available classes and that we can localise the point at which the object is located.

The simplest form of classification is to threshold the map. In a binary classification problem, a threshold is chosen (often zero) and all points with a value greater than or equal to zero are classed as object points, and all points less than zero are classed as background points. Figure 2.10 shows the thresholded version of the output shown in Figure 2.9. The regions that remain after this operation are passed to the final blob analysis stage where they are analysed and localised.

As we have a continuous map of classification, it is of course possible to do much more than just a binary threshold. Some previous work has looked at strategies for using this continuous output for performing multi-class classification [124, 127, 107, 106].

2.5.4.4 Blob Analysis

After thresholding, we have a binary map of points from which to try and calculate fitness. At the end of this stage, we ideally want a list of x,y points at which this detector (coupled with the feature set it is using) “thinks” that there are objects. This is then compared with the ground truth list of the actual object points. This is a hard task, and one which has seen many different approaches in the literature.

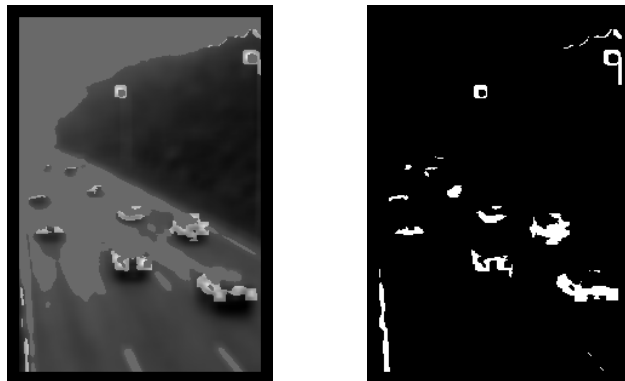


Figure 2.10: The image on the right is a thresholded version of the image on the left, which is the result of the detector application shown previously in Figure 2.9. Note that the greyscale image was normalised for display purposes, so the zero level is actually shown by a light grey.

The simplest option to consider is that every single pixel that has been marked could be used as a “guess”. This method adds a lot of complexity to the learning task as even a small incorrect region could contain hundreds of pixels, which would heavily influence the fitness values as it would add hundreds of false positives. In effect, this method totally ignores the structure of the blobs, and is making an assumption that each object should be hit exactly once.

It is reasonable to assume that if you train a classifier to detect an object, then it will probably hit that object multiple times within its boundary. We can therefore assume that any connected regions (the blobs extracted in the previous stage) are multiple hits on the same target. Therefore, if we could reduce the blob down into a single point, then this one point could be used for comparisons rather than all of them. There are many ways to do this, such as using the centre of the blob’s bounding box [126], its centre of gravity [102], or by using morphological operations such as skeletonisation.

2.5.4.5 Feature Set Usage

Although the work of Zhang [124] was used here to illustrate the process of using a static feature classifier, there are a limitless number of possible feature sets, with the only real limitation being the computational time needed to calculate them. For this reason, many previous approaches in the literature have used these simple statistical features. Tackett [115] used 7 features including “large” and “small” square windows calculating mean and

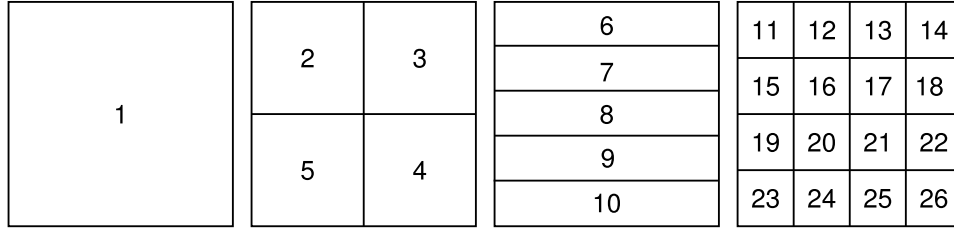


Figure 2.11: Feature set used by Winkeler et. al. [120]. Each zone is used for the calculation of both mean and standard deviation giving a total of 52 zones.

standard deviation around the point as well as global mean and standard deviations and local contrast.

In 1996, Daida et al. applied a GP system to the classification of low signal-to-noise ratio synthetic aperture radar (SAR) images for geoscience and remote sensing applications [22]. In their terminal set they used mean and gradient based features, such as a 5x5 Laplacian filtered version of a 3x3 Mean, a 5x5 Laplacian, a 3x3 Mean, and a 5x5 Mean. The choice of these features was probably influenced by the nature of their image data and the targets they were looking for which had properties in which gradients and grey-levels were important.

Whereas the work of Tackett and Daida used only a small number of features, some work, such as that of Winkeler [120] uses many more. The features used are shown in Figure 2.11. In all there are 52 features, as, for each zone, both mean and standard deviation is calculated. This is a very large number of terminals to use for any GP problem, especially when considering the fact that it is not even known if they are actually useful (in terms of solving the problem) or not. Many people use the argument that “GP will only use the terminals it needs” and to some extent this is true. However, it has long been known that having extraneous terminals in the terminal set will, at the very least, slow down the evolution and require more evaluations to be made as Koza has shown [55]. In problems like this where the data is so large and noisy, it is already very difficult to solve. Slowing down the evolution by having lots of “clutter” in the terminal set would make an already expensive process far more so. Koza’s experimental finding confirms the that the *curse of dimensionality* [6] applies to GP terminals and that as more features are added, the amount of training time grows exponentially.

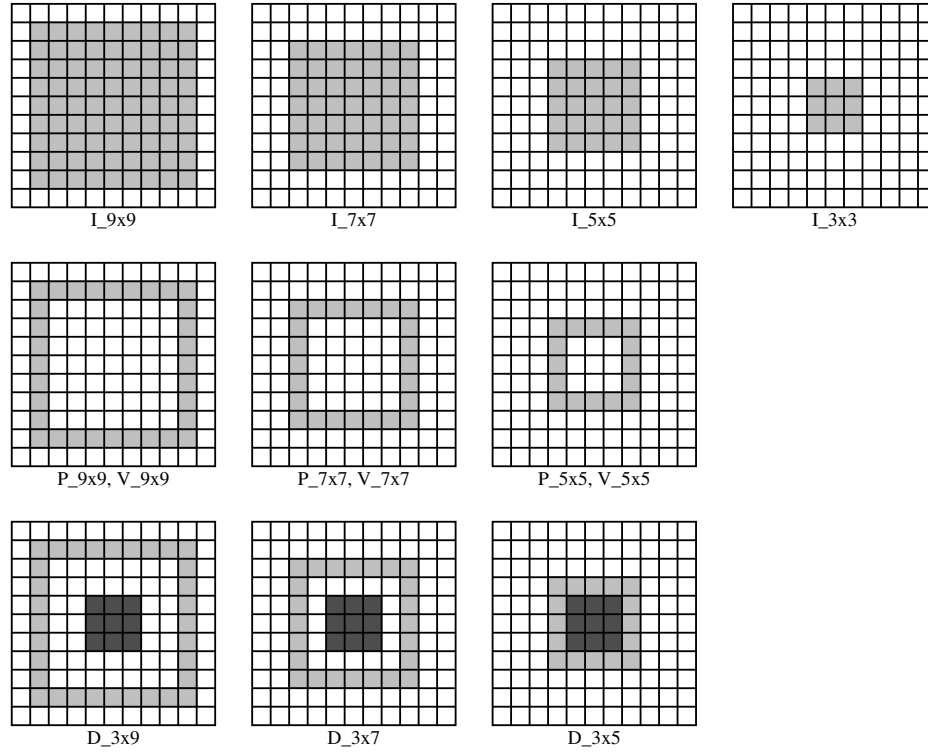


Figure 2.12: Feature set used by Howard et. al. [46]. The features $I_{N \times N}$ calculate the mean of the pixels in the $N \times N$ area, $P_{N \times N}$ calculates the mean of pixels on the perimeter of the $N \times N$ area, $V_{N \times N}$ calculates the variance of those same perimeter pixels, and $D_{N \times M}$ calculates the difference between the means of the $N \times N$ area and the $M \times M$ area.

Pixel statistics were used by Howard et. al. [46] to evolve ship detectors for use with SAR imagery. In this work, a set of 13 features was utilised, which calculated area means, perimeter means and variances, and the difference between perimeter means and area means. These features can be seen in detail in Figure 2.12. The approach produced well-performing detectors which achieved good results on unseen images. The authors also used the programmatic nature of GP in order to analyse and interpret the detectors produced. This is an important step if the detector is to be applied to a real world application.

The feature sets used in pixel statistic based classifiers are normally quite arbitrarily chosen. An exception to this would be the work of Howard et. al. [42] in which the effectiveness of several different feature sets are compared. Even though the comparison was the main focus of this work, the feature sets chosen were very innovative and powerful in their own right. The first feature set examined was based on pixel statistics, but they

were calculated in an interesting and effective way. Instead of looking at the statistics of rectangular zones, as many others have done, their features were based on the statistics of points lying in 4 concentric rings around the current pixel of interest. For each of these rings, they calculated four values –

1. the mean of all the pixel values on the ring
2. the standard deviation of all of the pixel values
3. the number of edges found on the ring
4. a measure of “edge distribution”, which quantified the spatial distribution of edges around the ring.

The values of these features were calculated at 4 different radii, in this case radii of 11, 22, 33 and 45 pixels were used. These feature are very powerful as they are *rotationally invariant* and would produce exactly the same values regardless of the orientation of the target object. This will make the classifier’s job significantly easier as it is fed consistent information about the target objects, rather than the confusing information that would be returned from the more common rotationally variant features.

The second feature set used was also based on the values around the perimeter of concentric rings. However, in this case the points were passed through a 1 dimensional⁴ discrete Fourier transform (DFT) [12]. The spectral coefficients produced are used as terminals. As the different sized rings produce different numbers of coefficients, only the lowest 7 frequency components were used. This decision may have thrown away some valuable data regarding edges in the larger rings though, and perhaps choosing a different set of components would have given better quality features.

The two feature sets were compared on the task of finding vehicles in aerial imagery. The experiments showed that whilst both sets performed well, the use of the DFT features produced better results in terms of detection, and also was more consistent (and therefore understandable) in the targets that it detected, which all looked similar. This work also used a novel multi-stage technique which will be discussed in more detail in Chapter 6.

⁴The transform is of course 1 dimensional as only the points on the ring are used, not the whole circular area. Traversing the edge of the ring produces the 1 dimensional signal.

2.5.5 Evolution of Features

The choice of feature sets for pixel statistic classifiers, whilst sometimes viewed as an arbitrary choice, can actually be seen as an optimisation problem in its own right. This is an interesting problem, but one that like so many others in this field is hampered by the huge computational requirements. Extracting features is itself expensive, so a population based optimisation will be many times more so.

Belpaeme [7] introduced a system for evolving sets of feature detectors based on a strongly typed GP architecture [73]. In this work, a single individual was actually a set of N GP trees which contained a mixture of image processing functions, coordinate based functions, and arithmetic functions (and used the three associated datatypes). The root node of each tree was constrained to produce a scalar result. During training a set of images was presented to the trees, however no specific task was used in this work, as the fitness function used was based on the producing outputs which tried to maximise the separation of the outputs in N dimensional space. This was calculated using information content measurements. Relating fitness to the spread of the outputs means that the set of features are implicitly co-operating, and increasing their diversity.

2.6 Chapter Summary

This chapter has presented an overview of the background knowledge necessary to appreciate the relevance and contribution of the rest of the work in this thesis. First the field of image understanding and computer vision was introduced, with particular emphasis on the task of object detection. Secondly, an overview of the field of evolutionary computation was given, introducing the major themes and algorithms involved. Genetic programming was described in detail. The sub-field of co-evolution was introduced and shown to have two major types, co-operative and competitive. Feature optimisation, including the subfields of feature subset selection, feature extraction and feature construction was also discussed. A summary of the use of evolutionary computation in image analysis tasks was given, demonstrating in particular the use of techniques that use pixel statistics as their inputs.

In the following chapter, we will see how this analysis of the field shows some interesting unexplored areas that could enable us to achieve high performance image analysis systems and open up some interesting questions about the nature of the learning process.

Chapter 3

Co-evolution of Features and Classifiers

In the previous chapter, we have seen how evolved classifiers based on pixel statistic features can be very effective. This chapter highlights some problems with the way in which the feature sets for these classifiers are chosen, and indeed problems with the task of choosing feature sets in general. We examine some previous work aimed at solving these problems, and propose a new method based on co-evolution.

3.1 Overview of Previous Approaches

Evolved object detection systems that use pixel statistic features can achieve reasonable results on a variety of problem domains, as shown in the previous chapter. However, while the classification part of the system is often carefully designed, the choice of feature set used is often a relatively unimportant detail with very little justification given for the choice made. In terms of the learning pipeline shown in Figure 2.5, only the second arrow, “Classification”, is being learned. Table 3.1 shows examples of feature sets described in the literature. It can be seen from diversity of the feature sets chosen, that there is little if any consensus about what makes a good feature set, or how many features should be used. For any given dataset, there is no way we can possibly know what will make a good quality feature set that will enable the classifier to solve the problem. Part of the reason for this is probably down to the fact that finding a good feature set for a particular

task is in itself a hard optimisation problem. This applies to both hand crafting feature sets or trying to automatically find a good feature set. It is so hard because there is no single best feature set and because the choice is intrinsically linked to the image data, the noise, and the classifier being used. All of these factors often lead to choices of large, overly general feature sets, or hand crafted, highly domain specific sets. As stated in the previous chapter, most work in the area of evolving detectors that use pixel statistic features has focused on the evolution of the detection algorithms and has not studied the choice of pixel statistics (except [125] which compared the performance of 3 different statistic sets, and [42] which compares simple statistics with DFT based ones).

3.2 Evolving Feature Construction

As there is no way to know what is a good feature set for a given problem, an obvious question springs to mind – as we have powerful evolutionary search techniques, why not apply them to the problem of finding good feature sets? This has been attempted, as described in section 2.4. However, this previous work just shifts the focus onto the preprocessor stage and uses deterministic classifiers in the detection stage. This is the opposite to the situation described at the start of this chapter where little or no thought was given to the preprocessor stage. With regard to the learning pipeline, only the first preprocessing stage is learned.

3.3 Combined Approaches

The next logical question that arises is whether both stages could be learned. This is quite difficult both conceptually and computationally. It is conceptually difficult as the detection stage must learn despite the fact that its inputs (the constructed features) are changing, and is computationally difficult as the time taken for learning the two stages is effectively multiplicative rather than additive.

In 1994, Andre[3] attempted to evolve both feature extractors and classifiers together. A hybrid representation was used in which each individual in the population consisted of a decision tree (being evolved using GP) and a set of feature detectors, which were represented as hit-and-miss transforms used in the field of mathematical morphology. The hit-and-miss transforms were represented as a ternary string and evolved using a GA.

Author	Statistic set
Tackett [115]	Means and standard deviations of small and large rectangular windows plus the global mean and standard deviation. A total of 7 zones.
Daida [22]	Pixel value, 3x3 area mean, 5x5 area mean, 5x5 Laplacian response, 5x5 Laplacian response of a 3x3 mean. 5 in total.
Howard [42]	Rotationally invariant statistics of 4 concentric rings. For each ring the statistics measured mean and standard deviation, number of edges and a measure of edge distribution. 16 in total.
Howard [46]	3x3, 5x5, 7x7 and 9x9 means, 5x5, 7x7, 9x9 perimeter means and variances, and the differences between a 3x3 area mean and 5x5, 7x7 and 9x9 perimeter means. 13 in total.
Winkeler [121]	Means and variances of 26 zones. Zone 1 was the entire 20x20 area, zones 2-5 were the four 10x10 quadrants, 6-10 were 5 20x4 horizontal strips, and 11-26 were 16 5x5 pixel squares. 52 in total.
Ross [103]	Features related to cross and plane polarised microscopy output. Angle of max. gradient, angle of max. position, max. colour during rotation, min. intensity, and min. colour. 9 in total.
Zhang [124]	Means and variances of 4 large and 4 small squares, plus the means and variances of 2 long and 2 short line profiles, passing horizontally and vertically through the origin. 20 in total.
Zhang [125]	Used 3 different terminals sets using means and variances of each zone. Set 1 used two different sized squares, set 2 had four concentric squares, and set 3 had 3 concentric circles. 4, 8, and 6 in total.

Table 3.1: Pixel statistic features used in previous work

These individuals were effectively GP trees with ADFs, except for the fact that the ADFs had a different representation to the main part of the tree. Andre tackled three binary optical character recognition (OCR) problems with this technique. The first of these was to evolve a solution that could distinguish between the letters L and T. The second was to identify which single digit (0-9) was shown. The third problem was to determine the 5 most similar examples to the one shown. During the evolution, either the main tree or the features were randomly selected and appropriate genetic operators were used for modification.

Although successful solutions were found for most of the problems, even for these very simple problems many hundreds of generations were necessary and many runs did not result in finding a successful solution. In some of the problems it also seems that there is a distinct bimodal distribution of successful runs, with easy problems being solved almost immediately in generation 0, but others taking at great deal longer if solved at all. This is an indication that the representation used may not allow sufficient flexibility to allow the trees or features to adapt to the problem. This may well be due to general problems with ADFs being too tightly coupled to allow flexible evolution. More discussion of this phenomenon can be found in Section 4.2.1.

Another hybrid system is that of Guarda et. al. [31] which was used to detect various facial features. This system attempted to learn both stages of the pipeline, one after the other. In this work the two stages were distinctly separate, with no feedback going between them. In the first stage of the system, a genetic algorithm was used to evolve 5x5 convolution masks which were the features of the system. These features were evaluated on the detection task in their own right, by convolving them with the input image and looking for their maximal responses. Their success at the task was used to calculate their fitness values. These types of feature are not powerful enough by themselves to solve the problem, but over a number of generations they did develop some abilities that could potentially be useful. The evolved features were added to a library which contained features from the current and previous experiments. The evolved features lacked the ability to use contextual, spatial information, so the second part of the system used genetic programming to evolve detectors which combined the “mediocre” features. Each detector is applied at every pixel in the image, and at each of these points can consider the responses of the

evolved features in each cell of a 3x3 grid around it. The detectors use a function set made up of fuzzy logic operations (and, or, not). An example detector might look for the presence of feature A above the point and not the presence of feature B to its left. This scheme allows the moderate performance of the features to be combined in a much more powerful way.

The system was tested on the task of eye tracking. One hundred images were used from a sequence, and training was performed on 10 percent of these. On the 90 test images, most eyes were correctly found, and as it is a tracking problem, the errors could be suppressed using Kalman filtering. However, it should be noted that as the images were taken from 100 consecutive frames from a video sequence, they would have been remarkably similar, and overfitting probably occurred. Each frame in the sequence showed the same person against a similar background, so this cannot be evaluated as a general eye-detector as it was trained with only one person. Results on a different unseen sequence were shown and the performance was considerably lower.

3.4 Co-evolution – A Possible Solution

These two examples highlight the problem that neither independence or tightly coupled co-operation are appropriate, and that a point somewhere between these two scenarios might be a better solution. In the first system the stages were linked together too tightly, effectively meaning that they were just single solutions. There was, therefore, too much co-dependence such that the stages could not explore and exploit the properties of their individual search spaces efficiently. In the second system the tight coupling was avoided, but at the expense of co-operation. The two stages were so separate that there was only minimal co-operation between them. In this case the system actually resembles a normal two stage system with a static feature set, apart from the fact that the feature set is evolved rather than human designed.

In this thesis it is proposed that a solution to this problem could be co-operative co-evolution [91], which allows us to create a system that has elements of both co-operation and separation. Using co-operative co-evolution (CC) allows the two stages (feature extraction and classification) to evolve separately in their own populations, thus allowing a thorough and wide search, but also be intrinsically linked in such a way that

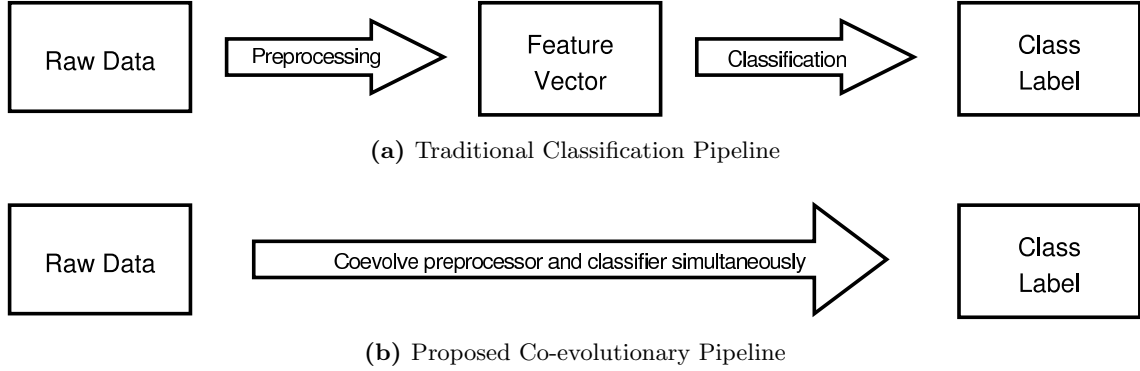


Figure 3.1: Comparison of the traditional and co-evolutionary classification pipelines

each part exerts a selective pressure on the other, causing both to strive towards a common goal. This should allow them to exploit each others strengths and compensate for each others weaknesses.

A co-evolutionary architecture such as this would be truly domain independent. The feature extractors could move and change their shape in order to more efficiently extract useful information from the raw data, whilst at the same time being implicitly pushed towards solutions which are useful to the classification phase. Total freedom (within the confines of the representation) can be given to the evolving feature extractors to allow them to adapt themselves to dataset, thus removing the need to have a human expert specify what may or may not be useful in solving the problem. The two stages effectively meld into one learning stage, and one in which no part of the system is fixed to a predetermined human-biased design.

At first this may seem like a multi-objective learning problem, but in multi-objective learning problems the performance of each objective can be measured. In this architecture however, each stage depends on the behaviour and output of the other and so the performance of individual subcomponents is undefined. Effectively, this means that we have not a multi-objective problem, but just one large single objective problem, that happens to include two parts. With regard to the classification pipeline (shown in the previous chapter but repeated here in Figure 3.1), whereas the traditional pipeline has two distinct stages of preprocessing and classification, a co-evolutionary architecture only has one operation, and that operation takes us straight from raw data to class labels.

Other techniques, such as neural networks, have been used in an attempt to create feature extractors and classifiers in a single stage. Each pixel of the input is fed directly into input nodes of the network and during training the hidden layers of the network learn useful features extractors, and further layers perform the classification. This scheme does however, suffer from severe scaling issues as the size of the network grows exponentially with the resolution of the input image. Successful approaches using this technique such as [66] only use very small resolutions (16x16 in this case). In general, this idea of using an input node per pixel of the image is not feasible for real-world problems as described by Blanz [10] -

...most research reported to date in which neural networks have been applied to machine vision focuses on explicit maps of the scene pixels “drawn” onto layers of network nodes so that each pixel in the scene corresponds to a separate node in the network layer. However, if “real world” image segmentation problems are addressed with connectionist classifiers one has to deal with images of 512 pixel by 512 pixels in size or even 2048 pixels per row and tens of thousands of pixels in a column for line scanning operations. This precludes the use of “neuron-per-pixel” architectures for economical reasons.

3.5 Chapter Summary

In this chapter the limitations of current methods of evolving pixel statistic based classifiers have been outlined. The conceptual framework of the classification pipeline has been used to illustrate how previous approaches to this task have either been lacking in some of the stages or have had too much or too little interaction and co-operation between the stages. Co-operative co-evolution was proposed as a possible way of avoiding all of these problems enabling us to co-evolve both pixel statistic based feature extractors and classifiers which use those features. This idea forms the basis of the rest of this thesis. The next chapter outlines an architecture that allows this to be achieved.

Chapter 4

A Framework for Co-evolving Features and Classifiers

As discussed in the previous chapter, co-evolution is a possible means by which we can simultaneously adapt both the feature extraction stage and the classifier learning stage of an object detection system. If the two stages were evolving co-operatively with each other, we could possibly attain more accurate and compact solutions. The fact that the feature extractors are also learning means that the need for domain knowledge is removed.

This chapter outlines a general framework in which to experiment with this simultaneous co-evolution of feature extractors and classifiers in object detection problems. First it describes the representation used for the evolution of feature extractors, and then describes the framework in which we can co-evolve the entire system.

4.1 Representation of Feature Extractors

As we saw in Section 2.5.4, “features” are often simply rectangular zones which, when overlaid on an image with the current pixel of interest as its origin, calculate some statistical properties of the pixels beneath them. This basic notion of features being rectangular zones which calculate pixel statistics is used in the creation of the co-evolutionary system. Each feature extractor will be made up of these rectangular zones and will be able to change position, size, shape and the statistic that it returns.

This may at first seem like a rather limited representation, but previous work, such as that shown in the previous chapter, has shown that these types of features can be remarkably powerful, even when only using simple statistics such as mean and variance/standard deviation. They also have another very attractive property – they can be evaluated very quickly in constant time, regardless of their size, as will be shown in detail in Section 4.1.4.

4.1.1 Basic Feature Properties

In order to apply an EA to a problem, we need to have a representation that can capture the essence of solutions to that problem and be adaptable in such a way as to thoroughly search the solution space. The proposed system uses a parametric representation of the properties of a feature extraction zone.

4.1.1.1 Zone Geometry

In the system, a single zone is represented by 7 real valued parameters which are summarised in Table 4.1. The first four of these parameters define the position (in polar coordinates) and the size (in pixels) of the zone. The origin of the zone’s coordinate system is always located at the current pixel of interest, so the zone positions are always relative to that point. Using polar coordinates allows a more natural adaptation of the zone’s position than cartesian coordinates would offer. This polar representation allows us to define the basic properties of the rectangle as shown in Figure 4.1. For a more intuitive interpretation, the distance r is measured to the centre of the zone.

4.1.1.2 Zone Shape

While the simple rectangular zone is quite powerful on its own, there are clearly many situations in which it will not be sufficient to solve the problem. The system introduces three other zone types which are just as simple in terms of the calculations they perform, in order to add extra flexibility to the system. All four zone shapes are shown in Table 4.2. All of these zone shapes can be made up from rectangles, which is important for reasons of efficiency outlined in Section 4.1.4.

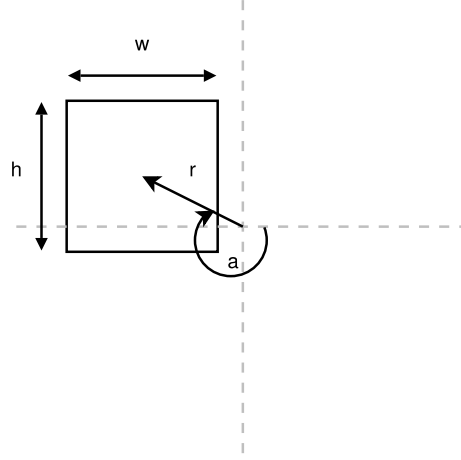


Figure 4.1: The general form of a feature extraction zone. Four parameters control the zone's position and dimensions. The origin of the zone is located at the current pixel of interest.

Parameter	Range	Description
Distance (r)	$[0, \infty)$	Distance of the centre of the zone from the origin
Angle (a)	$[-\pi, +\pi]$	Angle of the centre of the zone (in radians) from the positive x-axis
Width (w)	$[1, \infty)$	Width of the zone in pixels. Rounded to the nearest integer.
Height (h)	$[1, \infty)$	Height of the zone in pixels. Rounded to the nearest integer.
Shape	$[0, 3]$	Type of zone shape. Each value represents one of the zones described in 4.2
Orientation	$[-\pi, +\pi]$	Orientation of the zone. At present is only used to define the polarity of zone types 2 and 3.
Statistic	0 or 1	The statistic that zone types 0 and 1 calculate (either mean or standard deviation)

Table 4.1: Parameters that make up a single zone genotype



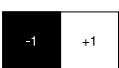

Shape	Structure	Description
0		The basic rectangular zone. Computes the statistic (either mean or standard deviation) of the pixels beneath it.
1		The ring shape. Similar to the rectangular zone except for a cut out. The size of the cut out is always one third of the size of the entire shape.
2		The horizontal weighted sum zone subtracts the sum of pixels on one side from the sum of pixel values on the other side.
3		The vertical weighted sum zone subtracts the sum of pixels on one half from the sum of pixel values on the other half.

Table 4.2: The four zone types used in the co-evolutionary system

4.1.2 Zone Presentation

Throughout this thesis, zones will be shown in two forms. The first is a textual form which simply lists the values of the seven parameters in the following way.

[*shape, distance, angle, width, height, orientation, statistic*]

For example, a 10x10 pixel “Ring” zone 5 pixels away from the origin calculating standard deviation would be shown as follows

[1, 5.0, 0.0, 10.0, 10.0, 0.0, 1]

Zones will also be shown in a graphical form which is generally far more intuitive. In the graphical form, the zones are shown relative to an origin which is the current pixel of interest. In this way the position, size, type and statistic of the zone can be instantly understood. All zones are shown with a 10 pixel “ruler” to show their scale accurately. The statistic that is returned by rectangular and ring zones are shown by the fill colour – mean zones are shown in blue and standard deviation zones are shown in red. Examples of both textual and graphical zone representations can be seen in Figure 4.2.

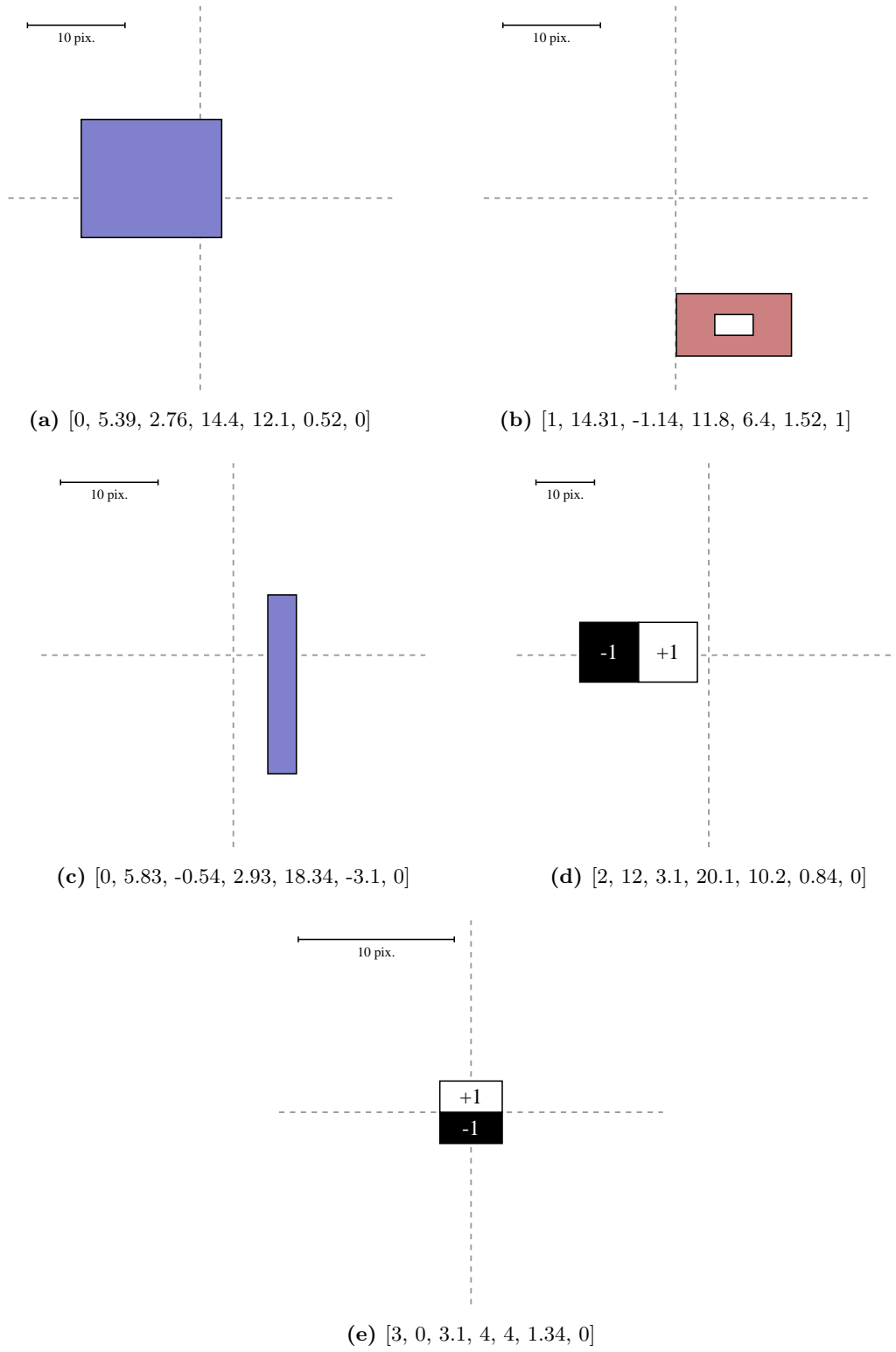


Figure 4.2: Some examples of zones with both their textual and graphical representations shown. Graphical representations always show a 10 pixel scale.

4.1.3 Evolutionary Operators

The operators that allow feature zones to be adapted act directly on the representation outlined in the previous section. Both crossover and mutation are used, with each returning one child.

4.1.3.1 Crossover

The crossover operator for this representation is quite simple. Given two parent zones, the position of the child zone will be randomly placed on the line joining the two parents, and the two polar coordinates will be modified to reflect this. The width and height of the child will be the average sizes of the two parent dimensions. The orientation of the zone is handled in a similar way. The shape and statistic parameters of the child are both random choices between the parameters of the two parent zones.

4.1.3.2 Mutation

Mutation of a feature zone involves a small change to one of the zone's seven parameters. If a zone is selected for mutation, one of the seven properties is then randomly chosen to be changed. Mutation affects each of the parameters in different ways, as outlined in Table 4.3.

Parameter	Description
Distance (r)	Addition of a gaussian random number (mean 0, $\sigma = r/10$)
Angle (a)	Addition of a gaussian random number (mean 0, $\sigma = 1$)
Width (w)	Addition of a gaussian random number (mean 0, $\sigma = width/2$)
Height (h)	Addition of a gaussian random number (mean 0, $\sigma = height/2$)
Shape	The zone will be changed randomly to one of the four shapes
Orientation	Addition of a gaussian random number (mean 0, $\sigma = 1$)
Statistic	Will be changed to the other statistic with probability 0.5

Table 4.3: The effect of mutation on each parameter

4.1.4 Evaluation of Rectangular Statistics

The fact that the feature zones are constantly evolving means that we cannot precalculate their outputs, which is always so beneficial in static zone systems. The co-evolutionary system will require the calculation of literally trillions of these statistical features during a single run, so the speed of their calculation is of paramount practical importance. A naïve approach (such as a standard convolution) to the calculation of these features would not be feasible, due to the sheer number of arithmetic operations required.

Ideally, a method is needed which can calculate the mean and standard deviation of arbitrarily sized image regions with constant (and small) time complexity. At first this seems like an unattainable goal, but it can be achieved using an alternate representation which has come to be known as the “integral image” [118].

4.1.4.1 Integral Image

In an integral image representation, we precompute an additional alternate version of the image (with the same dimensions as the original) in which each pixel contains the sum of all pixel values above and to the left of it. This is represented by equation 4.1, where SI is the integral Sum Image and $I(x, y)$ returns the pixel value in the original image.

$$SI(x, y) = \sum_{x' \leq x, y' \leq y} I(x', y') \quad (4.1)$$

With the image re-represented in this form, we can quickly compute the sum of pixel regions in the image using the simple relationship shown below, which is diagrammatically represented by Figure 4.3.

$$rectsum(x, y, w, h) = SI(x, y) + SI(x + w, y + h) - SI(x, y + h) - SI(x + w, y) \quad (4.2)$$

Using a similar method, we can also calculate the variance of a region in constant time. For this we need to precompute another version of the image, but this time store the sum of the squared pixel values. This new version of the image (which will be referred to as the Squared Integral SQI) shown in equation 4.3 can then be used to calculate the sum of squared pixel values $sumsq(x, y, w, h)$ using equation 4.4.

$$SQI(x, y) = \sum_{x' \leq x, y' \leq y} I(x', y')^2 \quad (4.3)$$

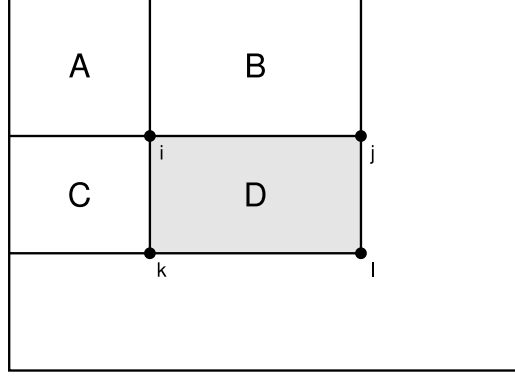


Figure 4.3: In the integral image representation, the sum of the pixels in rectangle D can be calculated using just four additions. The value at points i, j, k and l is the sum of pixels in A , $A + B$, $A + C$, and $A + B + C + D$ respectively. The sum in D is therefore $l - j - k + i$. Diagram modified from [71].

$$\text{sumsq}(x, y, w, h) = \text{SQI}(x, y) + \text{SQI}(x + w, y + h) - \text{SQI}(x, y + h) - \text{SQI}(x + w, y) \quad (4.4)$$

The *sumsq* function can then be used in conjunction with the *rectsum* function to calculate variance over an area using the relationship shown in equation 4.5.

$$\text{var}(x, y, w, h) = \left(\frac{\text{rectsum}(x, y, w, h)}{w \cdot h} \right)^2 - \frac{\text{SQI}(x, y, w, h)}{w \cdot h} \quad (4.5)$$

Calculation of Integral Images Although at first the precalculation needed for this approach may seem prohibitive, the integral image itself can actually be calculated very efficiently using only a single pass through the image. This is illustrated in figure 4.4, and equation 4.7 where $s(x, y)$ is the sum of the first x pixels on the current row. As the integral image is a cumulative representation, its completed parts can be used in the calculation of subsequent lines.

$$s(x, y) = s(x - 1, y) + i(x, y) \quad (4.6)$$

$$II(x, y) = II(x, y - 1) + s(x, y) \quad (4.7)$$

The use of integral image representations has exploded since the seminal work of Viola and Jones [118] in which they use simple rectangular statistic features to develop very

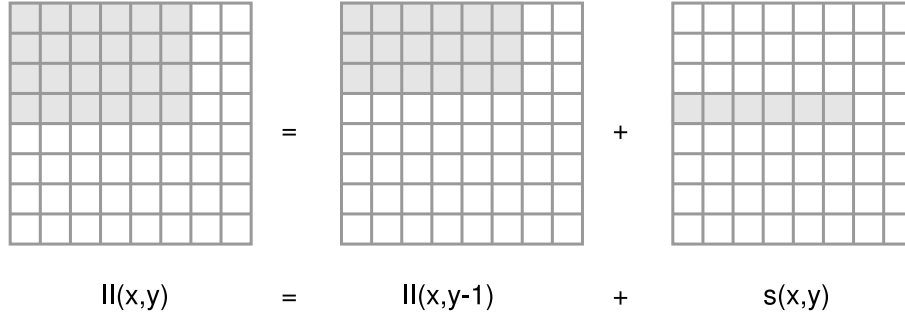


Figure 4.4: The integral image can be calculated using a single pass through the image, as it can use the partially formed integral image to calculate the rectangle above the current point ($II(x, y - 1)$) and a cumulative row value. This requires just three array lookups.

effective, and more importantly very fast, face detectors based on these features and a “cascade of classifiers”. The speed of the approach is largely down to their use of an integral image representation. The integral image representation was not particularly new and has been used for many years in the field of graphics (under different names such as “summed area tables” [20]) and was even used in early experiments with GP and images [84]. However, since Viola and Jones’ work, they are now a standard tool for this sort of task.

The two precomputed versions of the image SI and SQI need to be calculated once and then stored for the duration of the run. The dimensions of these two images are the same as the original, but evidently as this is a cumulative representation, the datatype used needs to be able to store numbers of several orders of magnitude larger than original (especially for the sum squared image). The total storage requirement can therefore be quite large if there are many images in a dataset and all need to be ready for fast access. However, this is a textbook case of where the benefits in terms of time complexity easily outweigh the cost of storing all of this data in memory.

4.2 Architecture of System

As stated earlier, the system will be co-evolving both feature extraction zones and GP based classifiers that will be using the extracted features. This is quite an ambitious goal as the classification stages will have to learn whilst its inputs are changing. The difficulty

of this task could be equated to trying to learn to type while every few minutes a keyboard designer swaps your keyboard for one which he thinks has a better layout.

The architecture of the system shown here attempts to sidestep these problems and provide a framework in which both stages of the system can co-evolve simultaneously without confusing the other stage too much. Indeed the very fact that they are learning in a changing environment, may make the final solutions more robust than they would be otherwise.

4.2.1 Overall Structure

The chosen design of the system is quite large and complex, and due to its co-evolutionary nature many things happen simultaneously. A helpful and interesting way to consider it is to think about the feature extractors as simply being functions of the GP classifiers. In GP the notion of an automatically defined function (ADF) is often used – each individual in the population has a number of functions attached to it which are evolving simultaneously with the main tree [55, 56]. In this way, module reuse can be utilised, often leading to smaller more efficient solutions. However, the ADF approach does have some drawbacks, one of which is that there is no explicit co-operation between ADFs so these totally independent functions may duplicate each other’s properties. Another similar problem is that fitness is only assigned to the whole program, rather than to its subcomponents, which means that badly performing ADFs could “hitch-hike” on the back of their well-performing colleagues. The opposite may also be true, and the main program may appear to be good, when in actual fact the ADFs are providing the only useful computation.

A solution to this problem was proposed by Ahluwalia [2, 1] in the form of a Co-evolutionary ADF (CO-ADF). In a CO-ADF system, an ADF is no longer statically linked to a particular individual but is instead represented as an entire subpopulation. At the time of evaluation, a complete solution is formed by combining one element from each population. Fitness is then assigned back to each subcomponent that formed the solution. In this way ADFs and main programs can evolve independently but co-operatively. Although the actual details are quite different, this principle is the inspiration for the framework proposed in this thesis for co-evolving feature extractors and classifiers, with the feature extractors each represented as independent subpopulations.

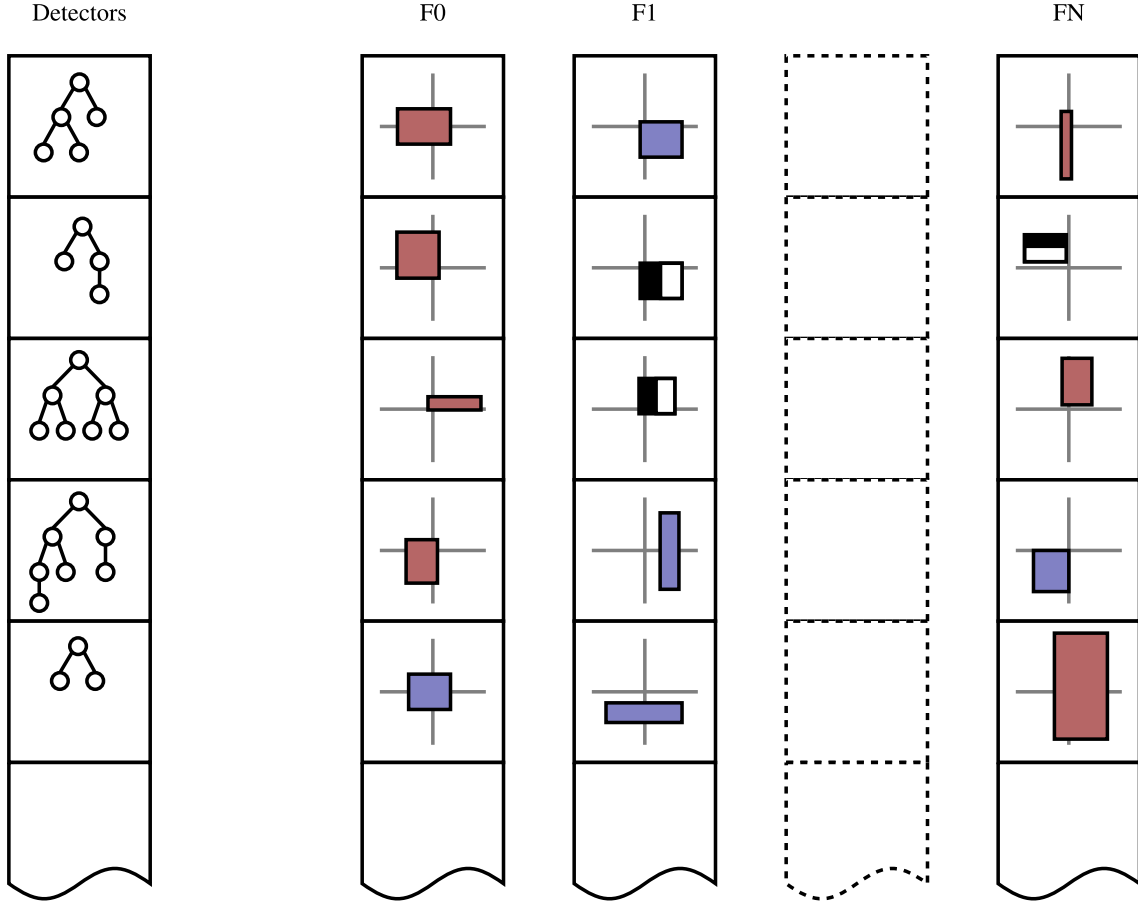


Figure 4.5: Diagram of the co-evolutionary framework, showing one detector population and a number of feature populations

This basic framework is shown in the diagram in Figure 4.5. There is one population of “standard” GP trees which behaves exactly as a normal GP population would. Importantly, the terminal set that these trees use, contains the feature terminals $\{f_0, f_1, \dots, f_n\}$, which represent the co-evolving features. f_0 would always represent the selected feature from the first population, f_1 is a feature from the second etc. All of the populations, whether they are classifiers or features, are completely independent of each other. Although they are co-evolving, they have no explicit knowledge of each other, and the only “communication” between them is implicit in the allocation of fitness, which tells them how well they perform when paired with individuals from other populations. The number of feature populations is fixed at a number supplied by the user. This is one of the few

parameters that stops the system being totally free of needing expert user interaction. See section 7.4.1 for some thoughts on how this constraint could be removed by having a dynamically changing number of zones.

4.2.2 Assembling Complete Solutions

Although the populations are separate in terms of their evolution, they are intrinsically linked in terms of their runtime behaviour. In fact, individuals in the detector population are totally meaningless without the feature populations. Conversely, the performance of members of the feature populations are only relevant in the context of the classifier population.

To actually perform an evaluation, a complete solution has to be formed from members of each population. In co-evolutionary terminology this is known as a collaboration. A collaboration in this system would take a detector and one feature from each of the feature populations. Once this detector/feature set pair is created it can then be evaluated using many of the same methods that were described in Section 2.5.4 for fixed feature set systems. After evaluation, the fitness values are assigned back to the individual components. The issues involved in collaboration and the subsequent credit assignment problem are discussed in the next section.

4.3 Collaborator selection issues

4.3.1 Selection Method

The selection of collaborators is a critically important issue to a system which relies on collaboration for such a large number of fitness evaluations. In co-evolutionary systems it is sometimes possible to evaluate components of the system independently and then use those scores to influence the selection of collaborators for assembling a complete solution. In this case, however, neither the performance of the feature extractor nor the classifiers can be assessed by anything other than a collaboration.

In order to assess how good a feature or classifier is, we would in the ideal case, evaluate *every* combination of classifiers and feature extractors and see how it performed

in all cases. This is clearly infeasible with multiple large populations as the computational complexity would rise exponentially.

The only other option is therefore to try and approximate this ideal case by performing a smaller number of random collaborations. The number of collaborations we pick in order to generate a single fitness score is known as the *collaborator pool size* [119], denoted here by the symbol p . We would like this number to be as high as possible (within computational limits) to approximate the ideal case.

In the system, this collaboration is achieved by iterating through the detector population (which is of size μ_d) and performing several collaborations with randomly selected features per detector. After evaluation, the detector and each feature that made up the solution adds the fitness value to a list, which is used later in the process to calculate final fitness values. This collaboration scheme can be shown more formally by the following pseudocode.

Algorithm 1: Collaboration selection scheme

- (1) **for** $d = 1$ **to** μ_d
- (2) **for** $c = 1$ **to** p
- (3) **foreach** feature population
- (4) select a random individual and add to feature set
- (5) evaluate detector[d] with feature set
- (6) assign fitness back to constituent parts

4.3.2 Credit assignment

Given that the sizes of the detector population and feature populations are μ_d and μ_f respectively, then after iterating through the detector population as described by Algorithm 1, each detector will have p fitness scores associated with it, and each feature zone will have approximately $\frac{\mu_d p}{\mu_f}$ scores attached to it. These scores represent how well each individual has performed in all of its pairings. At the end of each generation, we need to take this list of scores and convert it into a single fitness value which can be used in the selection process of that population's evolution.

There are three main ways in which we could possibly do this. The three methods are known as *optimistic*, *pessimistic*, and *hedge* [119]. They are described as follows.

optimistic In optimistic credit assignment, the best fitness score is chosen to be the final score.

pessimistic In pessimistic credit assignment, the worst fitness score is chosen to be the final score.

hedge In hedge credit assignment, the mean of all of the fitness scores is chosen to be the final score.

In this system, we are interested in finding features that can perform well with many different detectors and in collaboration with feature extractors from other populations. We are trying to find out the *potential* of each feature and in this case the notion of potential is best captured by using optimistic credit assignment. In hedge or pessimistic schemes, one bad score would be enough to drag down the score of an otherwise good individual which is quite the opposite of what is needed here. With large diverse populations, a well performing feature will undoubtedly be paired with bad detectors, or with bad features from other populations, and it would be unfair to judge it on performance of the other parts. We want one good score to be enough to guarantee the selection of the feature.

The situation for the detectors is similar – we want to judge them by their potential to perform well, and not be dragged down by one or two badly performing features. Optimistic credit assignment is therefore also used for the detector population.

4.4 Evolutionary Process Summary

In summary, the process of this framework is as follows. Firstly a single detector population is initialised. This population is a traditional GP population with each individual represented as a tree. The terminal set used in the trees contains a number of variables $\{f_0, f_1, \dots, f_n\}$, which are placeholders for the values computed by the evolving feature extractors. There are a fixed number of feature extractors available and each one evolves independently in its own population.

As each population represents only part of a complete solution, elements from each must be combined. This is largely a random process. For each detector, a number of

collaborations are tested. Each of these collaborations consists of the detector and one feature extractor from each feature extractor population. The collaboration is evaluated and the fitness value is stored in a list with each original constituent part of the solution. A number of these collaborations are performed for each detector.

At the end of a generation, every individual in each of the populations will have a list of fitness values representing the history of all of the collaborations that they have been a part of. A list of fitness values is not very useful for breeding purposes so optimistic credit assignment is then used to give each individual a single value – the best value in their list. Each population is then bred using the appropriate evolutionary operators.

The actual process of calculating fitness values is not relevant here as the same framework can be used with many different fitness functions. The framework simply provides a way of manipulating and breeding these populations. Chapters 5 and 6 will describe two very different systems which both use this common framework.

4.5 Detailed System Description

This section outlines some of the smaller details of the system. Although the actual implementation is less important than the overall system concept and architecture, some details are given here in order to achieve as full a description of the system as possible, so that it can be re-implemented by others.

4.5.1 GP Issues

The GP system used is based on the simple tree representation described earlier. After analysing off-the-shelf GP systems, it was decided that none met the requirements for this type of system, so a highly optimised GP system was written especially for this task. The system is written in C++ and is based on a “Virtual Function Tree” approach described by Keith and Martin [52] in their comparison of GP implementations. This is one of the fastest methods in their comparison, which makes it a good choice for this architecture due to the huge number of evaluations necessary.

As with many GP systems, the default behaviour and settings follow those described by Koza [55]. The trees are very basic, using no ADFs or other modular methods. They are randomly created using the “ramped-half-and-half” method which creates a variety

of shapes and sizes of trees up to a specified maximum depth. The *bloat* phenomenon is always a problem in tree based GP systems, causing the size of trees to increase exponentially as the run progresses. Traditionally this was limited by imposing a maximum depth that trees can grow to (a value of 17 is often used), or by adding a penalty value into the fitness function based on the tree's size. In this system a new and different approach has been chosen, which is called the *Tarpeian* bloat control method.

“Tarpeian” bloat control [89] removes the need to specify the maximum depth or size of a tree. Tarpeian bloat control is a clever compromise between tree growth and fitness that has a strong pressure towards smaller tree sizes, but does allow tree growth if it is accompanied by an increase in fitness. This is an ideal solution as sometimes complex problems require complex solutions, but in general we want the smallest solution possible. Although the Tarpeian method is based on rather complex theoretical observations, the actual implementation of the method is remarkably simple. Before evaluating a tree, its size is compared to the average tree size of the entire population. If it is bigger than the average, then with a given probability it is ignored, i.e. it is not evaluated and given a very bad fitness.

4.5.2 Parameters

There are a number of important parameters that govern the system. Each of these is described and discussed below. The actual values used for experiments, and other settings such as the function and terminal sets, are given in the relevant chapters.

Number of features N_f This parameter defines the size of the feature set used, which means that it also defines the number of feature populations in the system.

Detector Population Size μ_d and Feature Population Size μ_f As with any evolutionary computation problem, choosing an appropriate population size is somewhat of a “black-art”. In general the size of the population should be big enough that the solutions can represent the diversity present in the search space, so it is dependent on the complexity of the problem and the size and complexity of the function and terminal sets. In this

problem μ_f is often set quite low (around $O(50)$) whilst μ_d is set much higher (around $O(1000 - 2000)$).

Collaboration Pool Size p As described in section 4.3.1, this parameter determines how many random pairings we perform for each detector. Ideally this parameter should be set as large as possible so as to produce a better estimate of true performance. However, this parameter is one of the key multipliers in the run time of the algorithm, so for practical reasons is normally quite low, with a value of approximately 10 for most experiments in this work.

Genetic Operation Probabilities $cross_d, mut_d, clone_d, cross_f, mut_f, clone_f$ These parameters determine the probability of application of each of the three genetic operators. For each type of population, individuals can be crossed over, mutated, or cloned (directly copied to the next generation). Normally, only the crossover and mutation probabilities are specified as the figures sum to one, and so the cloning probability can be derived. The terms $cross_d, mut_d$ and $clone_d$ refer to the operator probabilities for the detector population, and $cross_f, mut_f$ and $clone_f$ refer to the feature populations. In the detector population, as with most tree-based GP applications, crossover is the dominant operator. In the feature populations, $cross_f$ is often lower, with mut_f being equally or more frequently applied.

Tournament Size $tourn_d$ and $tourn_f$ Tournament selection is used as the selection method for both types of population. As described in section 2.2, tournament selection selects the best individual from a small subset of the population as the parent. The size of this subset is given by the parameters $tourn_d$ and $tourn_f$ for the detector and feature populations respectively.

Tree Size Parameters $d_{create}, d_{mutant}, p_{tarp}$ The parameter d_{create} determines the maximum depth of newly created trees. Not all trees are created at this depth though, as the “ramped-half-and-half” method [55] is used, which creates a variety of shapes and sizes of trees up to that depth. The parameter d_{mutant} defines the maximum size of the randomly generated tree created during the mutation operation. The Tarpeian method

really only has one parameter, which we will call p_{tarp} . The probability of ignoring an above average size tree is $1/p_{tarp}$.

4.6 Chapter Summary

In this chapter a framework for the co-evolution of feature extractors and classifiers has been outlined. This framework consists of a number of co-evolving populations – one for the detectors and any number of populations for the feature extractors. A new representation was created to describe feature extractors made up of rectangular areas calculating local pixel statistics. Complete solutions are formed by combining individuals from each population. Fitness is assigned back to the components of those solutions and at the end of each generation the fitness scores allow us to make a judgement about the compatibility and performance of each individual, which produces their final fitness values. The populations can therefore co-evolve independently, but implicitly co-operate with each other. Whilst the idea of co-operative co-evolution is not new, this is the first time that a methods and representations have been created to allow the evolution of image features and object detectors in this way.

The one missing part of the system is the actual evaluation of the fitness of a complete solution. This is a complex task that can be performed in many ways depending on the problem at hand. Two methods of performing this evaluation for object detection tasks are shown in the next chapters.

Chapter 5

Using the Framework – Full Image Approach

In the previous chapter, a framework was described which facilitates the co-evolution of feature extractors and detectors which utilise those extracted features. This common framework could be used with many different evaluation mechanisms. This chapter describes a set of experiments using the framework with a novel fitness function that produces ingenious solutions, which offer both simplicity and accuracy.

5.1 Method Overview

The framework shown in the previous chapter creates “complete solutions” i.e. a detector and a feature set, which are evaluated to produce a fitness value that is then assigned back to the constituent parts of the solution. The actual calculation of fitness values clearly has to be based on the task itself i.e. on the comparison between the ground-truth data and the guesses that the evolved object detector makes.

In Section 2.5.4, the four stages of this process were outlined. First, the features were extracted, so that at every pixel of the input image we have a feature vector that can be used when the detector is applied at that point. In this detector application stage, we slide the detector over the image, instantiating the terminals from the feature vector and “running” the program tree. This gives a real value at each point in the image. As we are dealing with binary classification problems, this output is passed on to a third

stage where it is converted into a binary form by thresholding, producing a number of regions. Sometimes this produces very large regions. In this system these are discarded. Only regions smaller than 20% of the image are considered. Each of these regions is then converted into a single point normally by taking the centre point of the bounding box or the centre of mass.

This chapter describes a set of experiments using this form of *full-image* evaluation within the co-evolutionary framework. In order to do this, a fitness function is required which performs the four stages described above.

5.2 Fitness function

It is the job of the fitness function to take the list of points produced by the application of the detector and feature set, and produce a fitness value that meets the following two important criteria.

1. The fitness measure should express how good the solution is at solving the problem such that the fitness values could be used to rank an entire population in order of performance.
2. The fitness measure should also try and capture *how much* better one solution is than another. This enables the algorithm to move efficiently through the search space. It is important to capture enough detail as a more abstract measure might make it difficult to navigate the search space.

This section presents a methodology for achieving these goals in this particular domain of object detection. It can be seen from these requirements that fitness is a relative measure and it therefore makes little difference if the function requires maximising or minimising. The functions described in this section are all minimisation functions i.e. low values indicate better performance.

5.2.1 Motivation

In object detection problems the fitness function is often rather simplistic. When comparing an evolved solution to a ground truth image, the first step is often to count the number

of objects hit (true positives, or TP), the number of targets missed (false negatives, FN) and the number of incorrect guesses (false positives, FP). Some naïve approaches use a fitness function which just tries to minimise the raw measures of FN and FP with the ideal case being where $FN + FP = 0$ i.e. every target is hit and there are no false positives.

Although this meets the criteria of reflecting the quality of the solution, for many cases it does not give us enough information to make a useful judgement about how the individuals could be ranked. For example, in Figure 5.1 the output of two detectors is shown for the task of detecting the spiral pasta shapes. Each cross on the image represents a guess made by the detectors. It can be seen that the first detector has correctly hit some of the targets and has only one false positive, giving it a fitness value of 4. The second detector has missed all five targets and also has five false positives, giving it a fitness value of 10, making it seemingly worse in fitness terms. However, it is clear that the second of the two detectors has much better performance, as its false positive guesses are closer to the targets and with only a few small changes to the solution it could probably hit them all. This is a better solution in terms of its location in the search space – it is closer to a good solution, and it is therefore more likely that its offspring will search a better area of the space. A fitness function should reflect this fact.

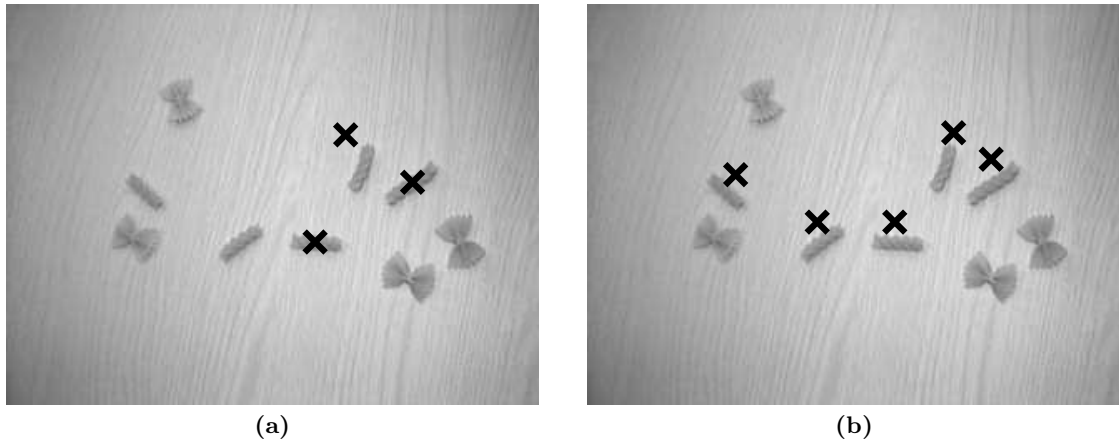


Figure 5.1: If we were minimising $FP + FN$ then solution (a) would have a fitness of $FP + FN = 1 + 3 = 4$ and (b) would have a fitness of $FP + FN = 5 + 5 = 10$. Even though this would point to (a) being much better, solution (b) is actually far closer to the optimal solution, so the fitness function is therefore incorrect.

In light of the deficiencies of the FP/FN based fitness function, we propose a new style of fitness function which factors in the distances between guesses and targets. We can now define the optimal performance of an object detector as follows.

- The detector should hit each target once (i.e. $FN = 0$)
- The distance between a guess and the centre of a target should be minimised.
- False positives should be minimised.

If these principles are obeyed and implemented in the fitness function, the scenario presented in Figure 5.1 can be avoided and the second solution would be correctly classified as being superior.

5.2.2 Implementation

The above rules can be implemented using a distance based fitness function. The approach taken here is to calculate fitness on a per-target basis – that is we calculate the fitness of all of the guesses around each target independently and then average each of these measures to produce a total fitness. The assumption behind this approach is that the guesses around any particular target are actually aimed at this target rather than any other.

Consider a target t which is a member of the set of all targets in an image T . The first step in the fitness function is to determine which guesses are around t i.e. which guesses are closer to t than to any other target. We can calculate this trivially by creating a proximity lookup map at startup time, which for every pixel in the image, indicates which target the point is nearest. An example of a lookup map can be seen in Figures 5.2(b) and (e). We can use a similar technique to create a distance lookup map, which stores the exact distance from any pixel to its nearest target. Examples can be seen in Figures 5.2(c) and (f). This precalculation avoids the repetition of expensive distance calculation during training, and has reasonable space requirements.

With this information easily accessible the fitness function can be implemented as follows. If a detector produces a set of guesses G then for each target $t \in T$ we can (using the proximity map) work out the subset of guesses that are closer to t than to any other target, G_t . For each of these guesses, $g_i \in G_t$, we can find out its distance from the target

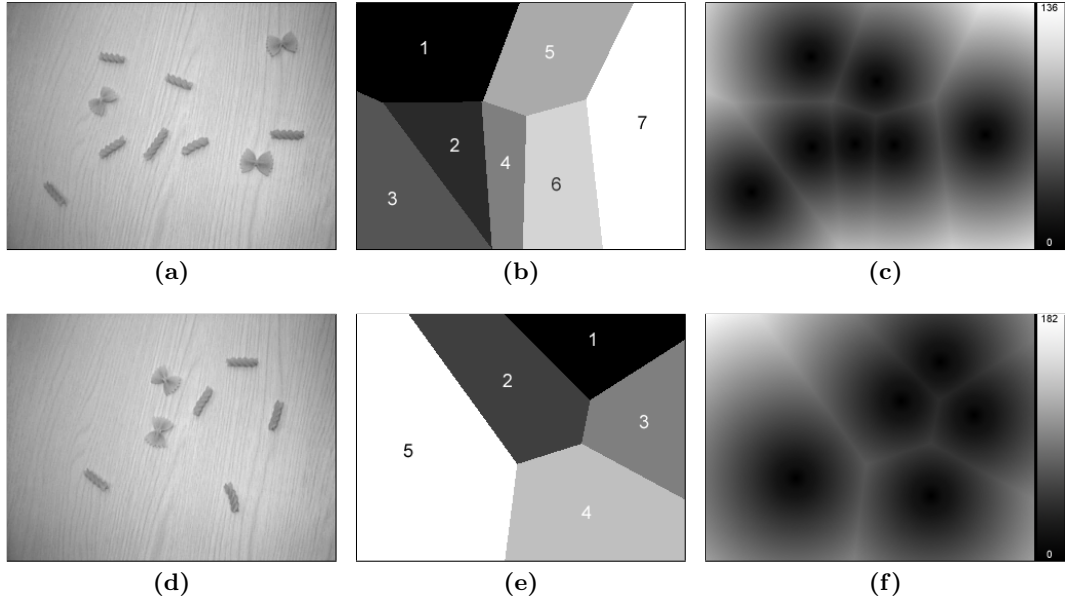


Figure 5.2: (a) and (d) show the two original images. (b) and (e) show the resulting proximity maps and (c) and (f) show the distance maps. These maps can be calculated once before training and cached. The proximity map indicates which target any pixel is closest to. The distance map provides the actual distance to that target (scale in pixels shown on right).

using the distance map. So in its most basic form we have the following fitness function which sums the distances of all points to their nearest target.

$$\begin{aligned}
 fitness(G) &= \frac{1}{|T|} \sum targetTerm(t) \\
 targetTerm(t) &= \sum_{g \in G_t} dist(g, t)
 \end{aligned} \tag{5.1}$$

This basic function meets the criteria of minimising distance between guesses and targets. However, it does not meet some of the other criteria such as that of minimising false positives and only making one guess per target. We can deal with this by adding another term to the function which multiplies the distance by the square of the size of the set of guesses $|G_t|^2$. This adds a strong pressure towards having as few guesses per target as possible, which has the side effect of minimising the number of incorrect guesses at a

target (i.e. false positives). The function therefore becomes

$$\begin{aligned} fitness(G) &= \frac{1}{|T|} \sum targetTerm(t) \\ targetTerm(t) &= \sum_{g \in G_t} dist(g, t) (|G_t|^2) \end{aligned} \quad (5.2)$$

5.2.3 Penalty Function

The last requirement of the function is that it should minimise false negatives (misses). Currently the function shown in Equation 5.2 does not exert any pressure towards solutions which do this as it is calculated on a per target basis and so targets with no guesses around them would incorrectly receive a perfect score of zero. So, a method is required to penalise any targets which do not have any guesses associated with them. Creating a penalty factor requires us to consider how bad it is to miss a target. This is a problem specific task. In some problems, missing a target could be very serious (for example in medical applications) whereas in other it would not be so serious. The approach taken here is to try and quantify what the penalty should be by comparing it to cases in the normal fitness function. For example we might say that “Missing a target would be as bad as guessing 3 false positives 50 pixels away from a target”. Expressing penalties in this way allows us intuitively convey the quality of the solutions and represent these intuitive ideas in values that are meaningful in terms of the fitness function and search space. In general we can define a penalty in terms of getting $count_{pen}$ false positives $dist_{pen}$ pixels away from a target. The fitness function is therefore

$$\begin{aligned} fitness(G) &= \frac{1}{|T|} \sum targetTerm(t) \\ targetTerm(t) &= \begin{cases} \sum_{g \in G_t} dist_{g,t} \times |G_t|^2 & \text{if } |G_t| > 0 \\ dist_{pen}(count_{pen}^2) & \text{otherwise} \end{cases} \end{aligned} \quad (5.3)$$

This penalty term is constant, and could be represented as such. However, in situations like this is it often more helpful to represent it in terms similar to the rest of the function so that the penalty can be thought about in a more intuitive way.

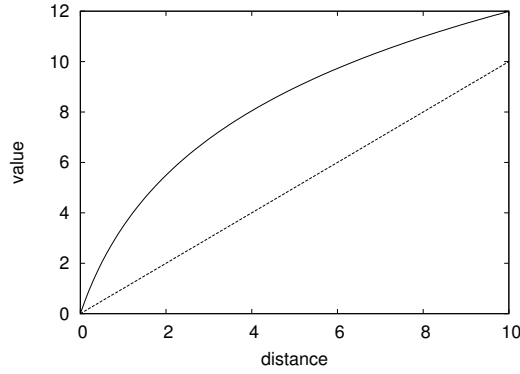


Figure 5.3: An example log weighted distance function. The gradient is steeper closer to 0 meaning that the search will be more easily able to focus in on the target.

5.2.4 A Non-Linear Distance Function

In order to allow the fitness function to converge more quickly to good solutions we can change the linear distance function to one which weights the path to the centre of the object differently to allow more sensitivity when closer to the target. This can be achieved by taking the logarithm of the distance (after adding 1.0 to avoid negatives) as shown in Figure 5.3. The steeper gradient closer to the target means that it will be easier to make improvements when closer to the target. This type of technique is often used in evolutionary computation as it is often fairly easy for the search to find a good general area, but is harder to home in on the exact minima. For example, in GP, a measure called *standardised fitness* is often used to rescale fitness values using the function $f = 1/(1 + x)$ which produces a similar curve to the log function proposed here. After adding the log function and a scaling parameter, α , the fitness function is as follows.

$$\begin{aligned}
 fitness(G) &= \frac{1}{|T|} \sum targetTerm(t) \\
 targetTerm(t) &= \begin{cases} \sum_{g \in G_t} \alpha \times \ln(dist_{g,t}) \times |G_t|^2 & \text{if } |G_t| > 0 \\ \alpha \times \ln(dist_{pen})(count_{pen}^2) & \text{otherwise} \end{cases} \quad (5.4)
 \end{aligned}$$

5.3 Figure of Merit (FOM)

In order to achieve the necessary evolutionary dynamics, the fitness function is highly non-linear. Although this is useful for the evolution, the values produced are not appropriate for human interpretation of the quality of a solution. A further measure is used called the *figure of merit* (FOM) which shows the performance of a solution in one simple figure. This is the same measure as used in [46], and is calculated as shown in Equation 5.5.

$$FOM = \frac{TP}{|N_t| + FP} = \frac{TP}{TP + FN + FP} \quad (5.5)$$

The FOM is calculated as the ratio between the number of true positives and the number of targets plus false positives. If $TP = N_t$ and $FP = 0$ then the FOM would equal 1.0. If $TP < N_t$ or $FP > 0$ then the FOM would move towards 0.0.

5.4 Experimental Method

A set of experiments were performed using the co-evolutionary framework and the full-image fitness function described in the previous section.

5.4.1 Overview

The experimental method used can be described as follows.

1. The dataset is divided into a training set and an unseen test set.
2. Before training starts, the following are calculated and cached for each image.
 - (a) A proximity map (as described in Section 5.2.2).
 - (b) A distance map (as described in Section 5.2.2).
 - (c) A summed pixel integral image (as described in Section 4.1.4.1).
 - (d) A sum squared integral image (as described in Section 4.1.4.1).
3. The single detector population and a user specified number of feature populations are created and randomly initialised.

Parameter	Value
Maximum tree creation depth, d_{create}	6
Maximum mutant creation depth, d_{mut}	4
Tarpeian constant, p_{tarp}	3
Detector tournament size, $tourn_d$	6
Feature tournament size, $tourn_f$	4
Detector population size, μ_d	1500
Feature population size, μ_f	100
Detector crossover probability, $cross_d$	0.7
Detector mutation probability, mut_d	0.15
Feature crossover probability, $cross_f$	0.35
Feature mutation probability, mut_f	0.65
Maximum generations, G	40

Table 5.1: GP parameters used for the full-image experiments

4. A number of collaborations between detectors and features are chosen as described in Section 4.2
5. For each collaboration, fitness is calculated by extracting the features at every pixel in an image and running the detector. These outputs are converted into points using the techniques described in subsection 2.5.4. Equation 5.4 is then used to produce a fitness value. This is done for each training image and the fitness values averaged.
6. At the end of the generation, final fitness values are calculated as described in Section 4.3.2
7. The populations are bred independently and the process is repeated until a suitable solution is found or the generation limit is reached.

Parameters common to all experiments are shown in Table 5.1.

5.4.2 Function and Terminal Sets

The terminal set for these tasks includes the variables $\{f_0, f_1, \dots, f_n\}$ which represent the feature values. The only other terminal node is the random constant node, which generates a new random scalar value in the range $-1 \leq \text{value} \leq 1$ on creation that stays with the node until it is deleted.

With the long term aim of producing fast object detectors in mind, a very simple function set was chosen, consisting of basic arithmetic and conditional functions. The four basic arithmetic functions are used – addition, subtraction, multiplication and protected division. The protected division performs a normal division except for the case in which the divisor is zero, when it instead returns a value of 1.0. Three conditional functions are used in total. The first of these are min and max, which return the minimum and maximum of their two inputs. The third conditional operator is an if-less-than-else operator (IFLTE). This operator has four arguments, the first two of which are compared using a less-than operator. If the first branch is less than the second, then the value of the third branch is returned as the result, otherwise the value of the fourth branch is returned.

5.5 Datasets

This section introduces the datasets that will be used to test the performance of the system. Some datasets have been artificially created to test various aspects of the system such as the ability to cope with different scales or orientations of the targets. The datasets have abbreviated names which describe them and are explained throughout the following sections. The grey level values of all of the images are scaled to lie between 0 and 1.

5.5.1 Synthetic Images

5.5.1.1 Simple Images – Set-T

The first dataset is a set of images which contain an extremely simple object detection task. The dataset consists of 100 images with a resolution of 300x300 pixels. Each image has a black background and contains between 10 and 20 white triangles with a radius¹ of

¹The polygon drawing routine is based on polar coordinates, so “radius” refers to the distance between the centre of the triangle and it’s vertices.

10 pixels. The triangles are at random positions in the image, with the constraints that they cannot be within 20 pixels of the image borders, and that they cannot be within 4 pixels of another triangle. This dataset is designated Set-T (triangles). The task is to find all of the triangles with minimal false positives. Examples of this dataset can be seen in Figure 5.4.

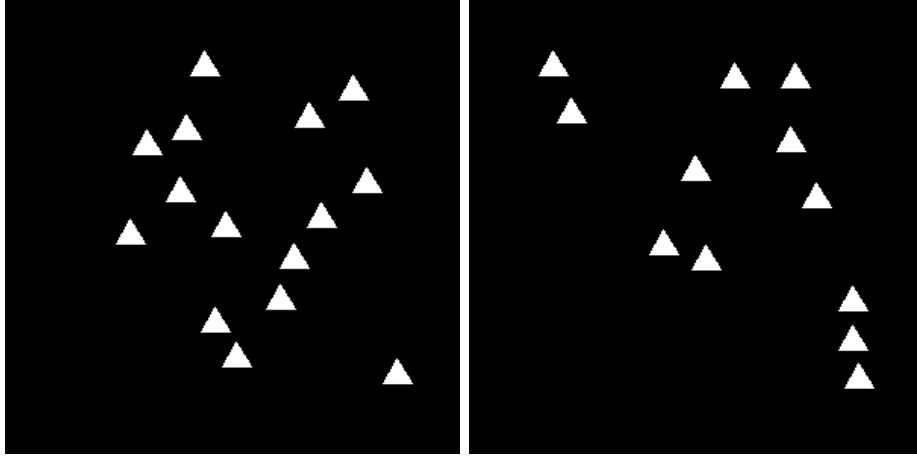


Figure 5.4: Two examples from the set-T dataset.

5.5.1.2 Simple Images With Noise – Set-TN

The second dataset is very similar to Set-T. It too has 100 images with 10-20 triangles in each. However, this set has Gaussian noise added to it to make the task more difficult. The noise has a standard deviation of 40 grey levels and a mean of 120. This noise is applied over the whole image, including the targets. This dataset is designated Set-TN (triangles with noise). Examples of this dataset can be seen in Figure 5.5.

5.5.1.3 Simple Images With Distractors – Set-TCN

The first two datasets (Set-T and Set-TN) only test the basic principle of the system, rather than its ability to find only one specific type of object. This third dataset adds distractor elements to the images. The distractors are white circles of radius 10. Again, there are 100 images in the dataset, each with a resolution of 300x300. The images contain between 10 and 20 objects, which have an equal probability of being triangles or circles. It is therefore possible for there to be no triangles present in an image. Noise is added

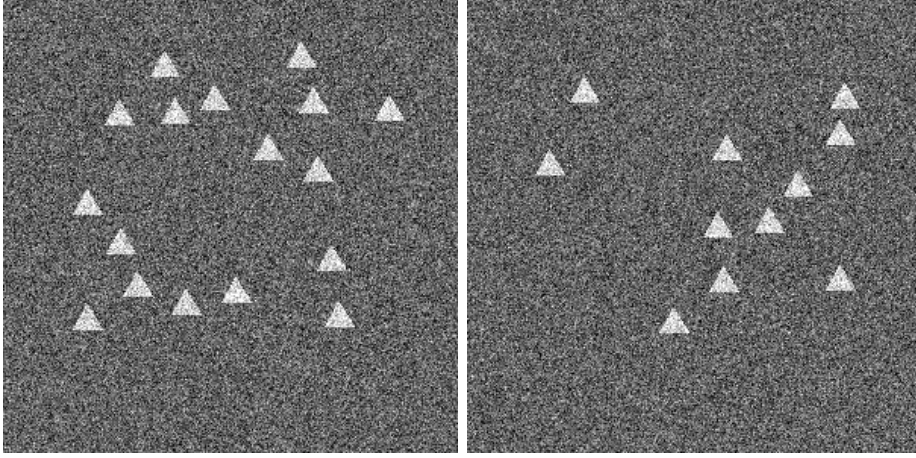


Figure 5.5: Two examples from the set-TN dataset.

to the same degree as in Set-TN. As before, the aim is to detect only the triangles. This will obviously be more difficult due to similarly coloured and sized, although differently shaped, distractor elements. This set is named Set-TCN (Triangles and Circles with Noise). Examples of this dataset can be seen in Figure 5.6.

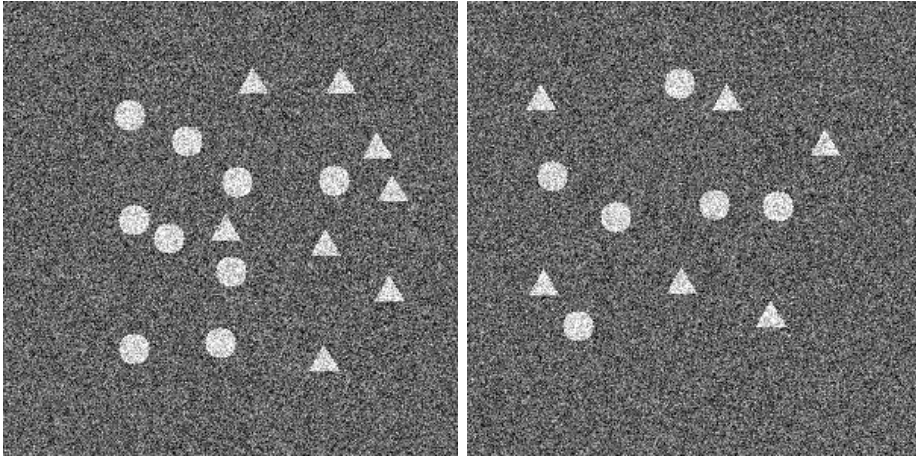


Figure 5.6: Two examples from the set-TCN dataset.

5.5.1.4 Scale Invariant Problems – Set-TCNS

It is often the case that objects may appear at different scales. This could either be due to an in-class variance in sizes (e.g. cars could be large or small), or due to distance or perspective effects. It is therefore important that an object detector can detect objects of

different sizes. This dataset attempts to test this ability by varying the sizes of the targets and the distractors. The radius of each object varies randomly between 10 and 20 pixels. Noise is present at the same levels as the previous sets. This set is named Set-TCNS (Triangles and Circles with Noise and Scale variance). Examples of this dataset can be seen in Figure 5.7.

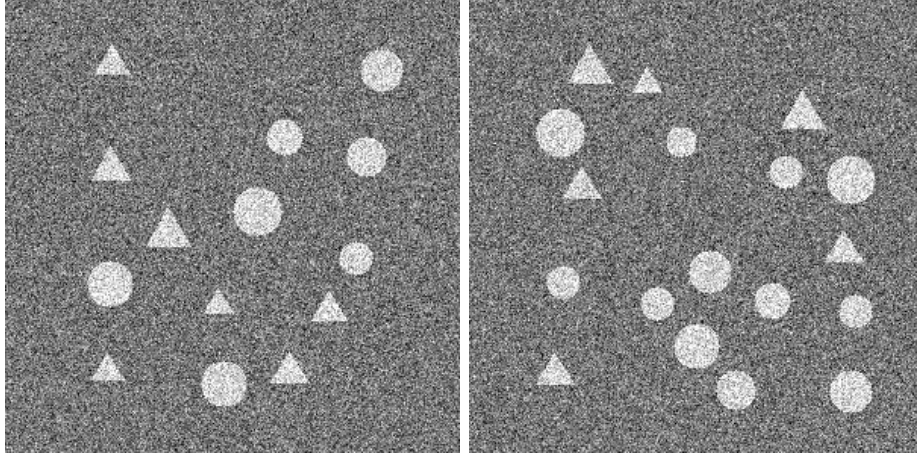


Figure 5.7: Two examples from the set-TCNS dataset.

5.5.1.5 Rotationally Invariant Problems – Set-TCNR

In the same way that scale invariance is an important property for object detection systems, so is rotational invariance. An object may appear at any orientation in the scene or due to the orientation of the viewpoint. This dataset allows each object to be rotated by an arbitrary angle. The size of the dataset, number of objects and noise levels are the same as previous datasets. This set is named Set-TCNR (Triangles and Circles with Noise and Rotational variance). Examples of this dataset can be seen in Figure 5.8.

5.5.1.6 Scale and Rotation Invariant Problems – Set-TCNSR

This dataset combines both scale and rotational variability. Each object has a random scale and rotation. This set is named Set-TCNSR (Triangles and Circles with Noise and Scale and Rotational variance). Examples of this dataset can be seen in Figure 5.9.

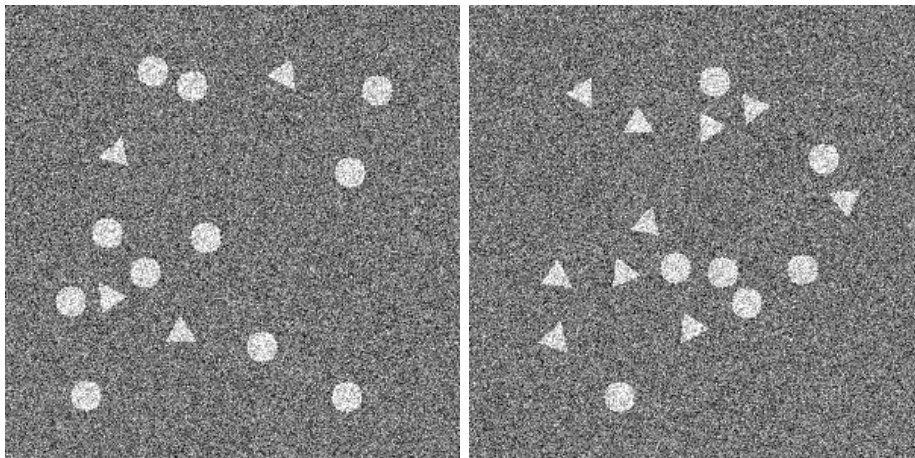


Figure 5.8: Two examples from the set-TCNR dataset.

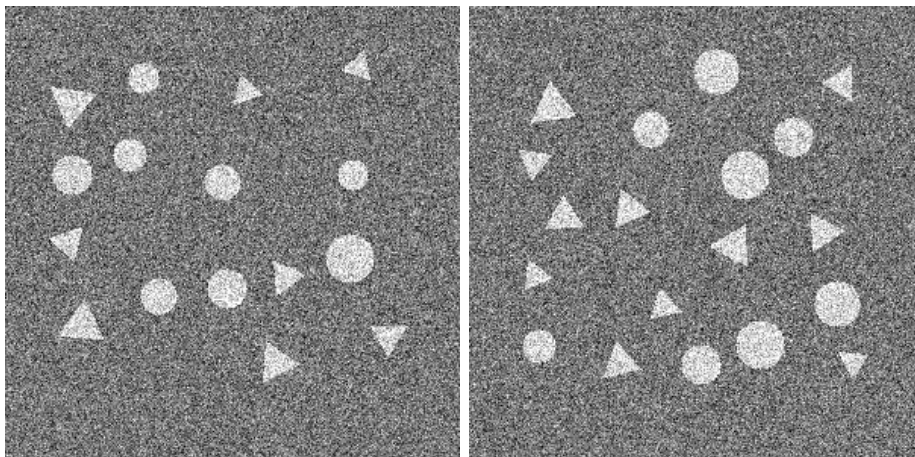


Figure 5.9: Two examples from the set-TCNSR dataset.

5.5.2 Real World Images

Although noise was used in the synthetic dataset, the images can not possibly provide the richness and diversity of data that is present in natural images. Several real-world datasets were also created to test the abilities of the system, and although some of them may not be truly useful real-world applications, they provide useful insight into the capabilities of the co-evolutionary system.

5.5.2.1 Keys and Coins – Set-KC

As its name implies, the keys and coins dataset contains images of keys and coins. There are 100 greyscale images. Each image shows a random number of silver keys and silver coins at random orientations. Two different sizes of coins were used. The background of the image is moderately noisy and unevenly lit. The keys and coins were chosen as they have similar colour and reflectance, so a detector would have to learn to identify the shape. Examples of set-KC can be seen in Figure 5.10

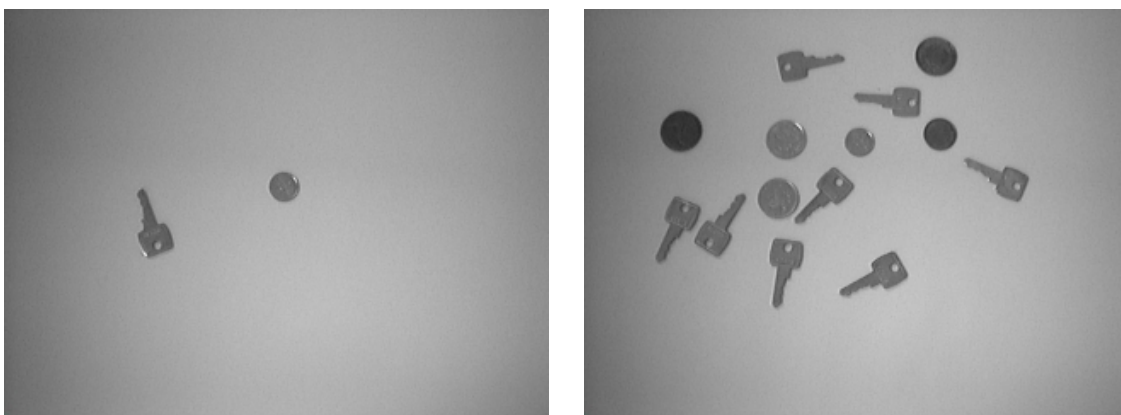


Figure 5.10: Two examples from the set-KC dataset.

5.5.2.2 Pasta Dataset – Set-P

In order to provide more complicated shapes than the circles and coins used in previous datasets, a dataset was created using two types of dried pasta – spiral shapes and bow-tie shapes (more precisely called *Fusilli* and *Farfalle*). There are 100 320x240 images in this dataset, with each image containing a random number of each type of pasta which can be at any orientation. The different types of pasta clearly have the same colour and reflectance properties, which forces the system to use shape information instead of these properties. The task is to detect only the spiral shapes. The background is far more complex than Set-KC. It has a wood-grain texture and is unevenly lit. Examples of Set-P can be seen in Figure 5.11.

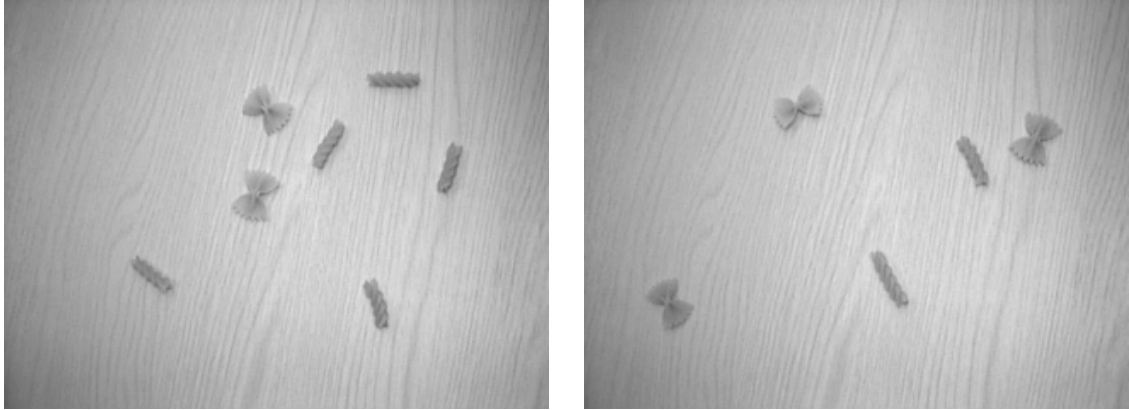


Figure 5.11: Two examples from the set-P dataset.

5.5.2.3 Aerial Traffic Images – Set-AT

The final dataset in the series is a difficult aerial imagery problem. This dataset consists of 20 images of the road network and car parks around the British Columbia Institute of Technology, which were cropped from the original high-resolution ortho-rectified photograph². Twenty smaller images were cropped from the original and each is of a different size. There are many cars in each image, each of which covers an area of approximately 300 pixels. In the examples shown in Figure 5.12 it can be seen that the images are highly cluttered, with many distracting elements and shadows. The cars can appear at any orientation and are of different shapes, sizes and colours. This is a truly difficult object detection problem – in fact during the manual marking-up stage to create the ground truth, many of the cars were initially missed and some false positives were introduced. The Aerial Traffic dataset is designated Set-AT.

5.6 Experiments

Experiments were conducted on the training images from each of the datasets. The performance on the unseen test set was also monitored at each generation to assess generalisation performance, but this was of course *not* fed back into the system. The following sections describe the *best* solutions (in terms of performance on the test set) found over the 20 runs on each dataset to indicate the peak levels of performance attained on each problem.

²Image copyright Selkirk Remote Sensing Ltd. <http://www.selkirk.com/>. Reprinted with permission.

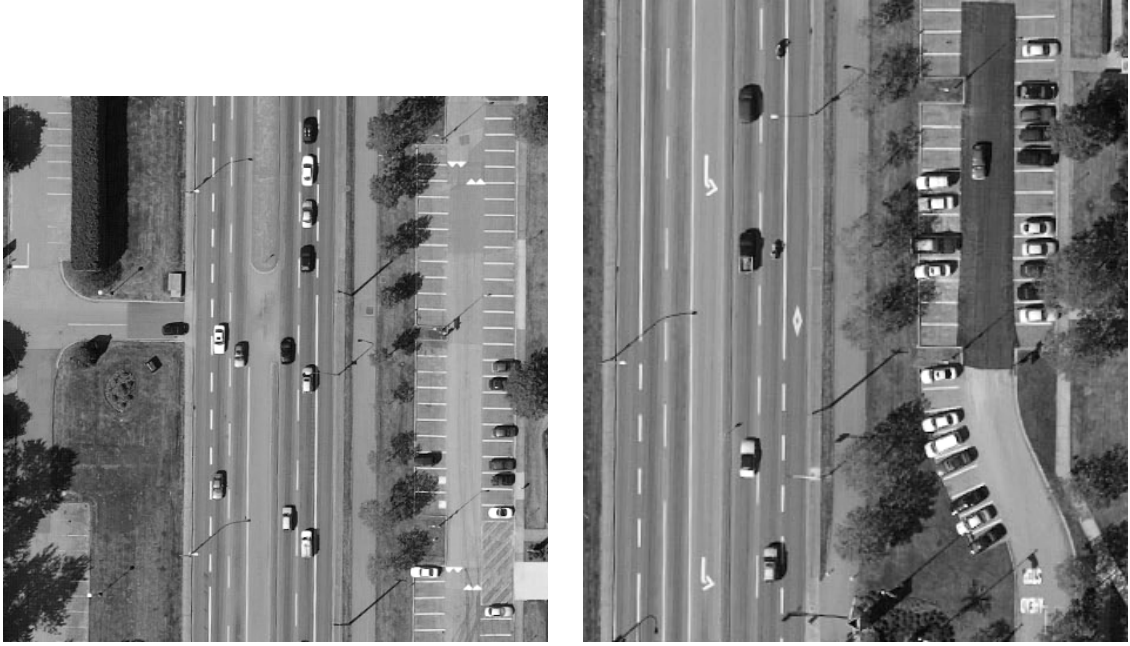


Figure 5.12: Two examples from the Set-AT dataset.

Multiple runs of each experiment were performed to assess the general performance of the system (rather than just the best instance) and these results are discussed in Section 5.7.

For each experiment, the solution (the detector and the zone set) is shown. The detector is shown as a simple tree and the zones are shown in graphical form as described in section 4.1.2. Also, for each solution, a sample output is shown. This shows the stages of the solution on one of the *unseen test images*. The original image is shown, along with the raw output of the detector, the thresholded output, and the final “guesses” that this produces. The guesses are colour coded to illustrate what type of point they are. True positive points are shown in red, and false positives are shown in blue. If a target is missed totally, it is highlighted in yellow.

5.6.1 Experiments on Set-T

The system was run on the simple image dataset consisting of white triangles on a plain black background, Set-T. This is an extremely easy task and it comes as no surprise that the system was very quick to deal with this problem, and produced a perfect solution (fitness=1, FOM=1) after just two generations. Figure 5.13 shows the solution (the tree

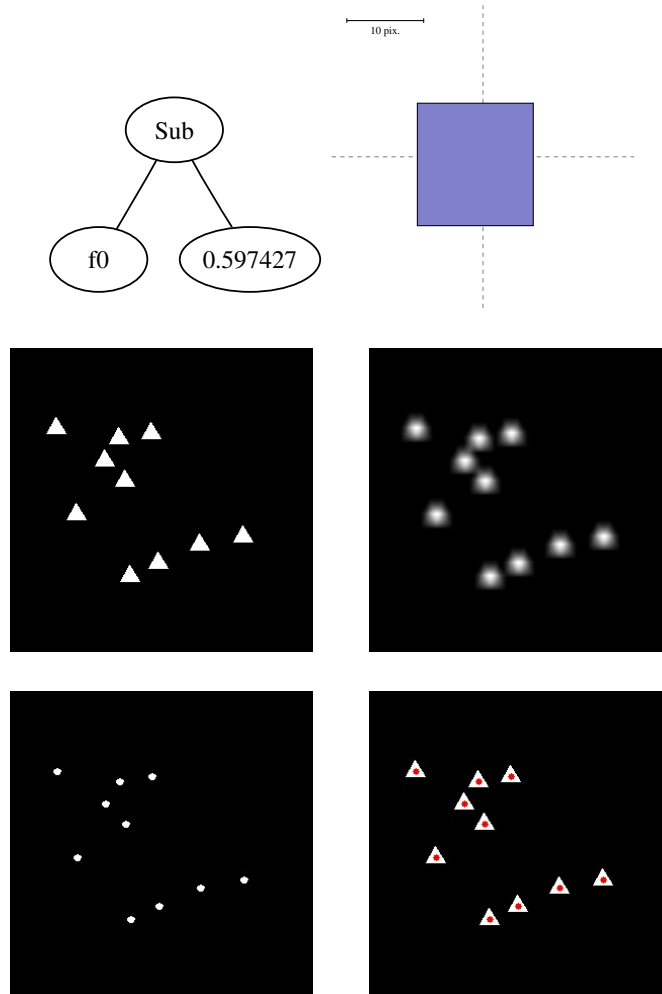


Figure 5.13: Best solution from 20 runs and sample results for set-T. The tree and features used are shown (top) along with sample output. Top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses.

and the feature extractors) as well as a set of example images showing the input image, the greyscale output of the detector, and the thresholded image. Looking at the solution shown in Figure 5.13 it is immediately clear how it works. The single feature calculates the mean of the pixels in the 15x15 area approximately centred on the origin. It should be noted that this feature has evolved to totally encompass the 10 pixel radius of the triangles. The detector simply takes this value and subtracts 0.6. The subtraction of 0.6 means that when the image is thresholded at zero, only the very centres of the triangles

remain, ensuring that no regions can merge together, as would have been the case with a smaller value.

5.6.2 Experiments on Set-TN

The noisy triangle set proved to be slightly more difficult. Although a perfect solution in terms of the FOM was produced after just 3 generations, a perfect score in terms of fitness took a lot longer to achieve. The evolution quickly evolved the ability to hit all of the targets within the desired accuracy, but took longer to be able to hit the centres perfectly. The best solution is shown in Figure 5.14. This solution could be collapsed down into a simple subtraction, similar to the Set-T solution shown previously.

5.6.3 Experiments on Set-TCN

Finding the triangles accurately in spite of the circular distraction elements is obviously a harder task as the two elements are similar in every way except for their shape. For this reason the system was allowed to use up to 4 features. After 9 generations the system found a solution with FOM of 1 on both training and testing images. This solution did not totally minimise fitness, meaning that the guesses were not perfectly centred on the triangles. Figure 5.15 shows the solution. When thresholded at zero, the output is mostly white, but the solution has cleverly used the constraint that an object cannot be any larger than 20% of the image size, and so most of this white area is removed, leaving only the areas that were inside the triangles. Effectively the solution “opens” up the circles so that they “leak” into the background. The triangles remain closed, ensuring that areas inside them remain. This is a nice example of evolution using a subtle aspect of the problem in order to solve it more efficiently. Figure 5.15 also shows the final feature set overlayed over a triangle and a circle. As some areas of the feature extractors are over the background when centred on a triangle, it is clear that they would produce very different values for circles and triangles.

5.6.4 Experiments on Set-TCNS

On the scale invariant problem, Set-TCNS, the system took much longer to achieve good performance. On the training set a solution with a FOM of 1.0 was reached after 11

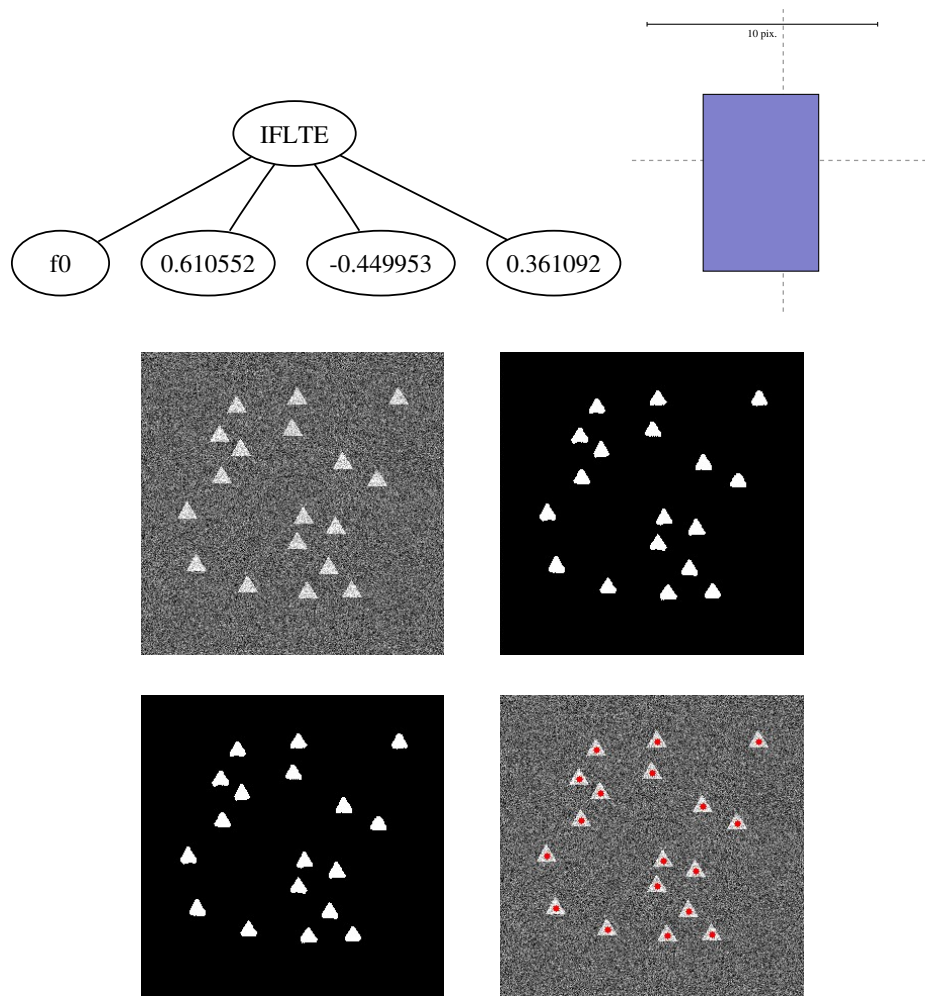


Figure 5.14: Best solution from 20 runs and sample results for set-TN. The tree and features used are shown (top) along with sample output. Top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses.

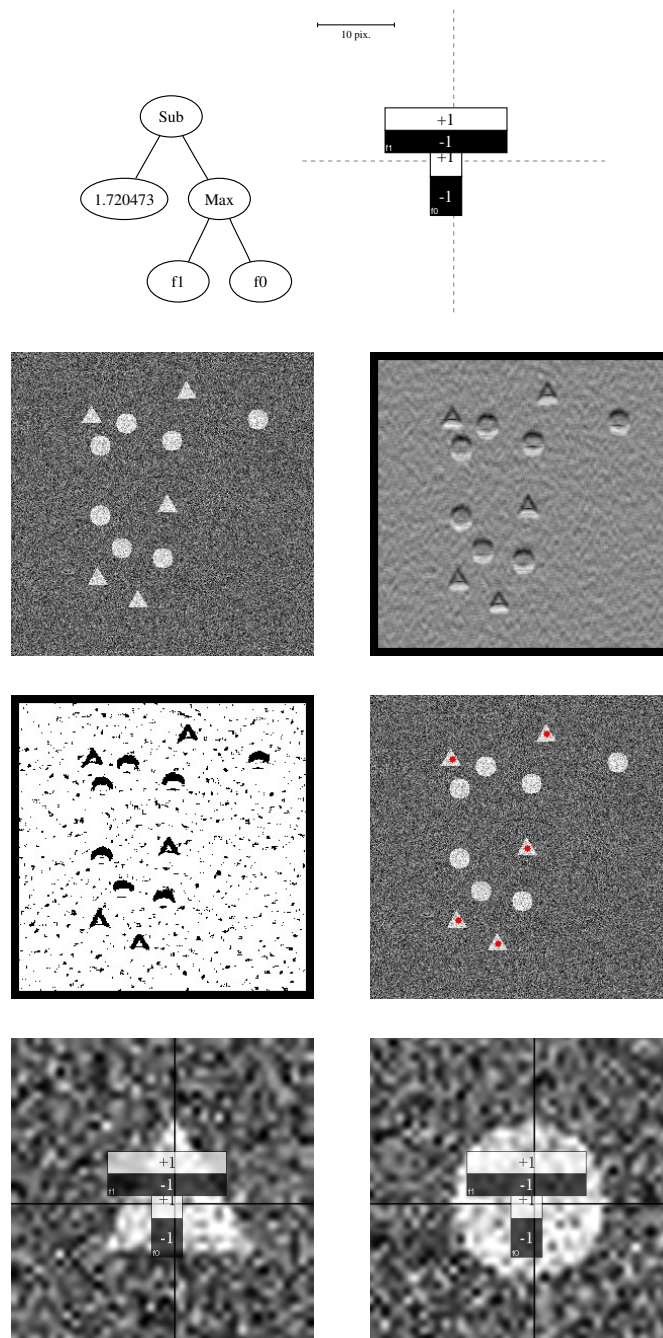


Figure 5.15: Best solution from 20 runs and sample results for set-TCN. The tree and features used are shown (top). The next four images show – top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses. The bottom images show the feature zones overlayed over some magnified triangles and circles. This gives insight into how it solves the problem.

generations, but it took a further 10 generations to achieve the same level of performance on the unseen test set. The solution produced has 11 nodes and uses three features. The features have adapted their sizes so that they encompass the largest shapes and the detector responds very strongly to the narrower top of the triangle. The solution and example responses can be seen in Figure 5.16.

5.6.5 Experiments on Set-TCNR

As would be expected, a solution for rotational invariance task proved to be harder to evolve. This is almost certainly due to the fact that the system is trying to classify objects with varying orientation using rectangular features which cannot themselves rotate. After 26 generations a 21 node solution was evolved which had a mean training set FOM of 1.0 and mean test set FOM of 0.974 (s.d. 0.055). In order to cope with the extra complexity of the task, the solution uses five features, although the influence of some of them is negligible. Although the test set performance is not as good as previous experiments, it is still very high and equates to just 4 errors over the dataset containing hundreds of targets. Further training after this generation resulted in overfitting indicated by a drop in test set performance and increase in solution size. The solution can be seen in Figure 5.17. Like the set-TCN solution, this solution also relies on the property of the system of rejecting large objects. The triangles are very effectively kept closed whilst the circles are opened out into the main body of the image and therefore ignored. On some of the testing images, the solution produced small numbers of false positives. An example of this can be seen in Figure 5.18. When the objects are close together the solution can sometimes blend the regions together which on some occasions leads to a closed area being formed in the “bridge” between the two regions which is subsequently marked as a guess. This sort of mistake is an indication of the variability present in the dataset relative to the previous sets. A larger training set may have included more examples of situations like this and helped to avoid this type of error.

5.6.6 Experiments on Set-TCNSR

The best solution produced for Set-TCNSR is one of the most ingenious ever produced by this system. It was evolved after 23 generations and uses just seven nodes (after pruning

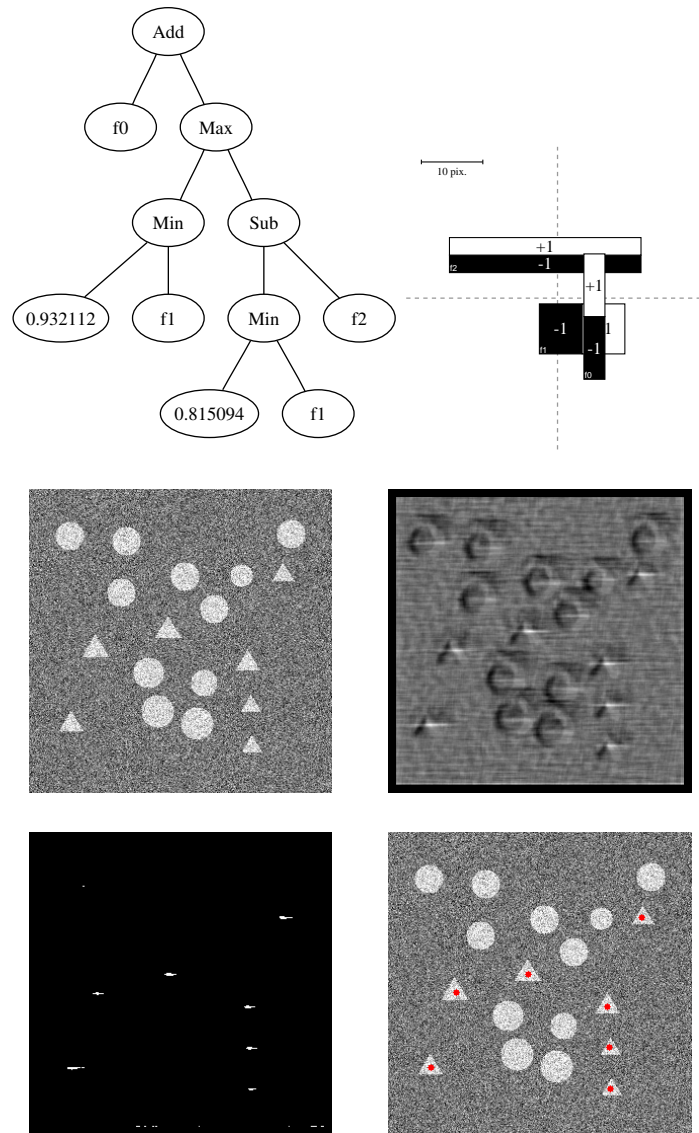


Figure 5.16: Best solution from 20 runs and sample results for set-TCNS. The tree and features used are shown (top) along with sample output. Top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses.

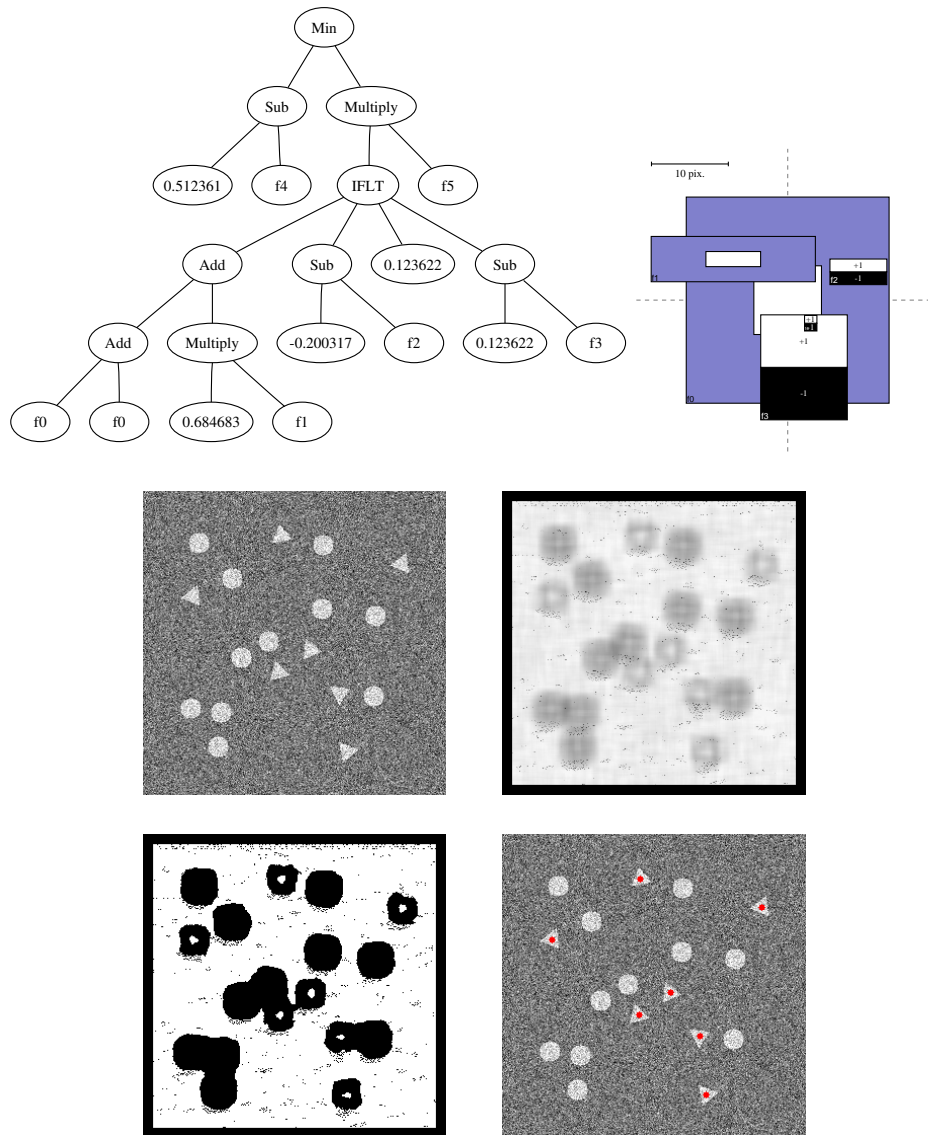


Figure 5.17: Best solution from 20 runs and sample results for set-TCNR. The tree and features used are shown (top) along with sample output. Top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses.

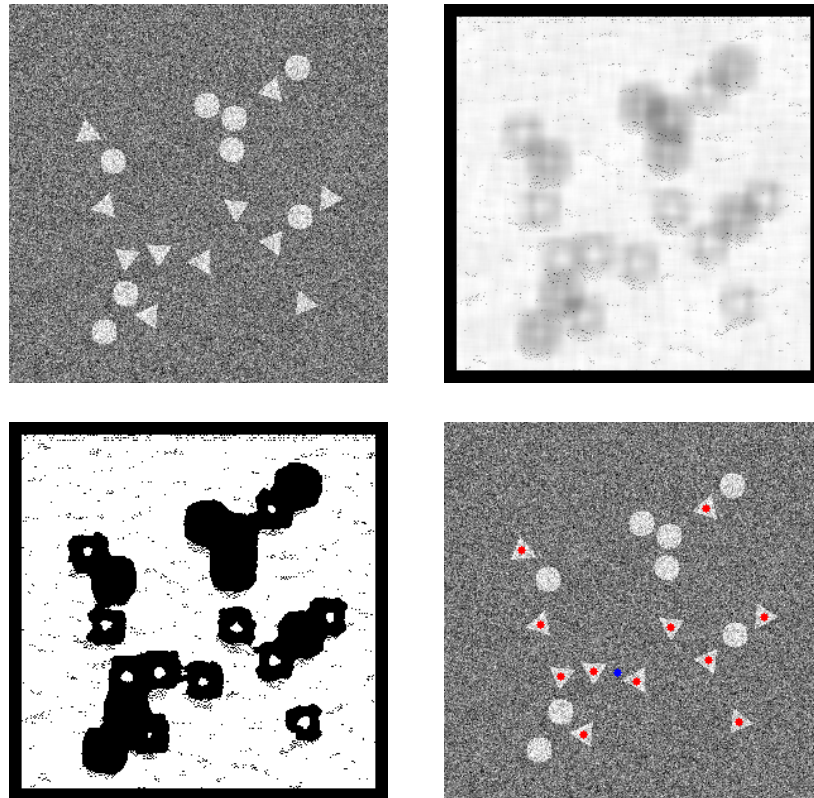


Figure 5.18: Example of the sort of mistake present in the Set-TCNR solution. Clockwise from top-left - the original image, the raw detector output, thresholded output, final colour coded guesses. False positives occur when there are two triangles very close to each other or a circle close to a triangle. The noise present causes the two to blend together and produce closed regions.

of nodes which have no functional effect) and two features. It scored a average FOM of 1.0 on the training set and achieved a FOM of 0.97 (s.d. 0.049) on the unseen test set³. The way in which it solves the problem exemplifies the way in which this technique can find the simplest possible solutions to seemingly complex problems by exploiting subtle aspects of the data. The solution uses two vertical weighted sum features with opposite polarities spaced evenly above and below the origin. The symmetry of a circle would make the two responses of these features cancel each other out. A triangle however, is guaranteed *never* to have equal areas above and below its centre and will therefore always produce a response from these features, regardless of its scale or orientation. This solution also illustrates the power of implicit co-operation - the two features were evolving in *independent* populations, but managed to form a complementary structure.

5.6.7 Experiments on Set-KC

The Set-KC problem proved to be deceptively easy for the system to solve. Although at first glance it looks to more difficult than the previous datasets the system consistently located a subtle feature that went unnoticed during the construction of the dataset. As stated in section 5.5.2.1, the colour and reflectance of the keys and coins are similar and the background is unevenly lit and at first it was thought that these aspects would challenge the system. However, after just two generations, the system found a perfect solution to the problem, not by using any complex analysis of the shapes, but by looking for the holes in the keys! The holes are distinguishing features of the keys that are always present and clearly visible, and of course no coin has them. The system found the simplest feature that it could exploit to solve the task and used this to its advantage. This is not something that most human designers would have thought of first.

The solution itself is also very interesting. It basically performs a threshold such that the holes become white and the key around them is darker. However, because of the uneven lighting, a simple constant threshold would not work over the whole image. Instead, the system has reinvented the *adaptive threshold* algorithm [17]. In an adaptive threshold, the value chosen to threshold at is not a constant but is instead based on the statistics of

³The solution is actually also better at solving Set-TCNR and Set-TCNS than the solution shown for those datasets and could be used instead in real applications.

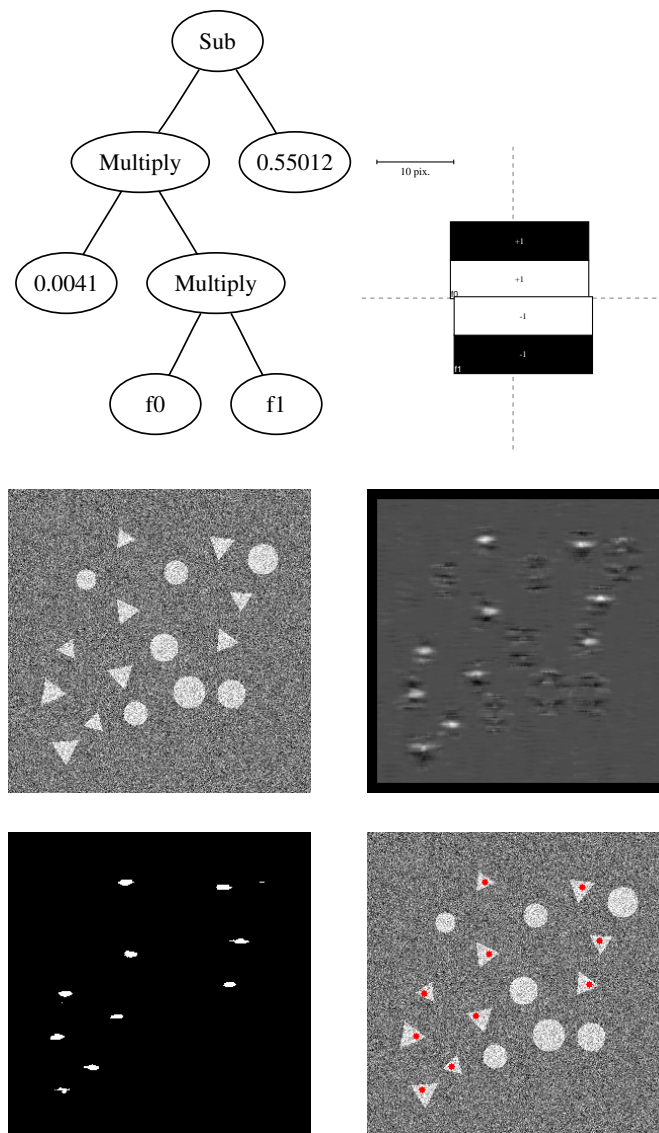


Figure 5.19: Best solution from 20 runs and sample results for set-TCNSR. The tree and features used are shown (top) along with sample output. Top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses.

the region around it. So, instead of saying "is this pixel lighter or darker than the global constant" it says "is this pixel light or dark relative to the local area". The adaptive threshold is an extremely useful algorithm and is widely used in real-world applications where uneven lighting is present.

The evolved solution implements the adaptive threshold by using two features. The first is a large feature (30x36 pixels), which calculates the mean of a large region around the origin. The second feature is a tiny (3x2 pixels) zone which calculates the mean of a small area around the origin. An IFLTE node is then used to perform the actual comparison, comparing the large area mean to the small area mean, and returning a high value if the small area value is higher than the large area average. The solution can be seen in Figure 5.20.

5.6.8 Experiments on Set-P

The best solution to Set-P was evolved after 19 generations of training. The solution contained 25 nodes and used only two zones. The run terminated after 19 generations as the solution achieved a perfect FOM of 1.0 on all training images. When applied to the unseen test set, the solution performed slightly worse with an average FOM of 0.98 (s.d. 0.06). The solution can be seen in Figure 5.21. This solution uses the same technique as previously shown solutions of creating a border around the objects, but allowing that border break in non target objects, making them "leak" into the background. This solution is remarkably simple considering the rotational variance of the objects and uneven lighting. The use of two zones calculating standard deviation effectively creates a multi-scale edge detector. The outputs of these detectors is combined to form the pasta detector. The tree could be further simplified by removing the IFLTE node - as it's first child node is such a small value, the comparison will almost always be false, and the IFLTE node will therefore always return $f1/0.947751$. As the parent node of the IFLTE node is finding the minimum of this and $f1/0.73956$ then this will always return $f1/0.947751$. At this point the tree can then be further simplified to produce the expression $\min(f0, 0.45188) - 2.375857761 * f1$. This tree can be seen in Figure 5.22. The reduced solution has less than one third of the amount operations as the original, with obvious consequences for run time, and when

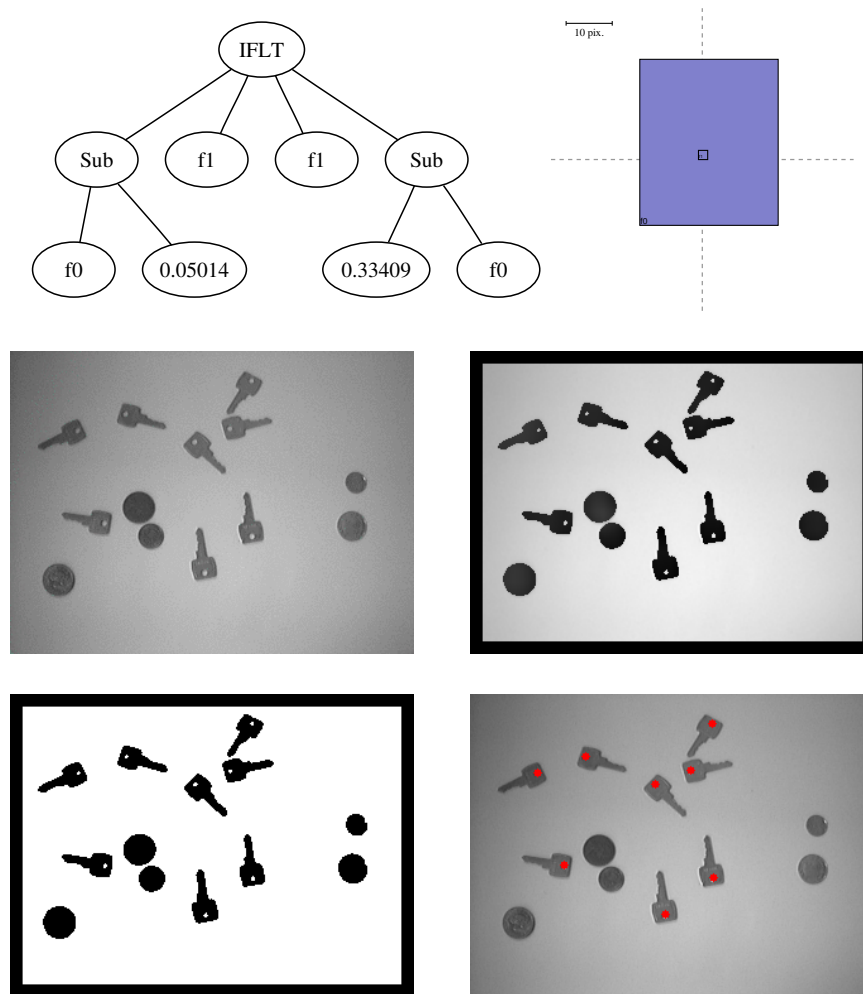


Figure 5.20: Best solution from 20 runs and sample results for set-KC. The tree and features used are shown (top) along with sample output. Top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses.

Dataset	Training set FOM	Test set FOM	Solution size	Features used	Generations (hours)	Training time
set-t	1.0 (0.0)	1.0 (0.0)	3.6 (1.1)	1.2 (0.42)	1.6 (0.61)	108 (44.05)
set-tn	1.0 (0.0)	1.0 (0.0)	3.65 (1.1)	1.7 (0.75)	2.8 (1.5)	196 (104.9)
set-tcn	1.0 (0.0)	1.00 (0.008)	4.9 (1.3)	2.3 (0.89)	11 (2.1)	789 (148.0)
set-tcns	1.0 (0.0)	0.99 (0.020)	23.4 (9.6)	2.4 (0.89)	12 (2.9)	990 (263.3)
set-tcnr	0.99 (0.046)	0.90 (0.056)	30.4 (6.8)	4.2 (1.0)	34 (4.2)	3046 (406.0)
set-tcnr	0.98 (0.031)	0.92 (0.054)	30.3 (11)	3.4 (0.61)	26 (11)	2308 (1020)
set-kc	1.0 (0.0)	1.00 (0.008)	13.1 (4.2)	2.3 (0.48)	2.2 (0.42)	162 (32.38)
set-p	0.99 (0.035)	0.94 (0.050)	35.5 (6.5)	2.7 (0.65)	27 (8.8)	2476 (853.9)
set-at	0.078 (0.020)	0.063 (0.021)	49.7 (13)	8.5 (2.9)	40 (0.0)	4308 (254.9)

Table 5.2: Summary of results for the full image method. All figures are averages over 20 runs. Standard deviations are shown in parentheses.

applied to the dataset produces exactly the same results as the original. This analytical approach to simplification could possibly be automated in future work.

5.6.9 Experiments on Set-AT

The performance on Set-AT was disappointing compared to the success on previous datasets. Over many runs, the system did not once create a solution with a FOM of more than 0.14. Most targets were missed and the cluttered background of the images produced many false positive points. An example of the best solution produced can be seen in Figure 5.23. The algorithm finds it very difficult to distinguish between cars and the cluttered background. On the training images, this solution hits an average of only 1.7% of all targets and produces an average of 10.7 false positives per image. The surprisingly poor performance on this dataset motivated the work describe in the next chapter.

5.7 Analysis

The previous section illustrated only the best solutions evolved using the full image methods on the datasets. Table 5.2 shows the mean performance (in terms of FOM on the test set) of the system over multiple runs.

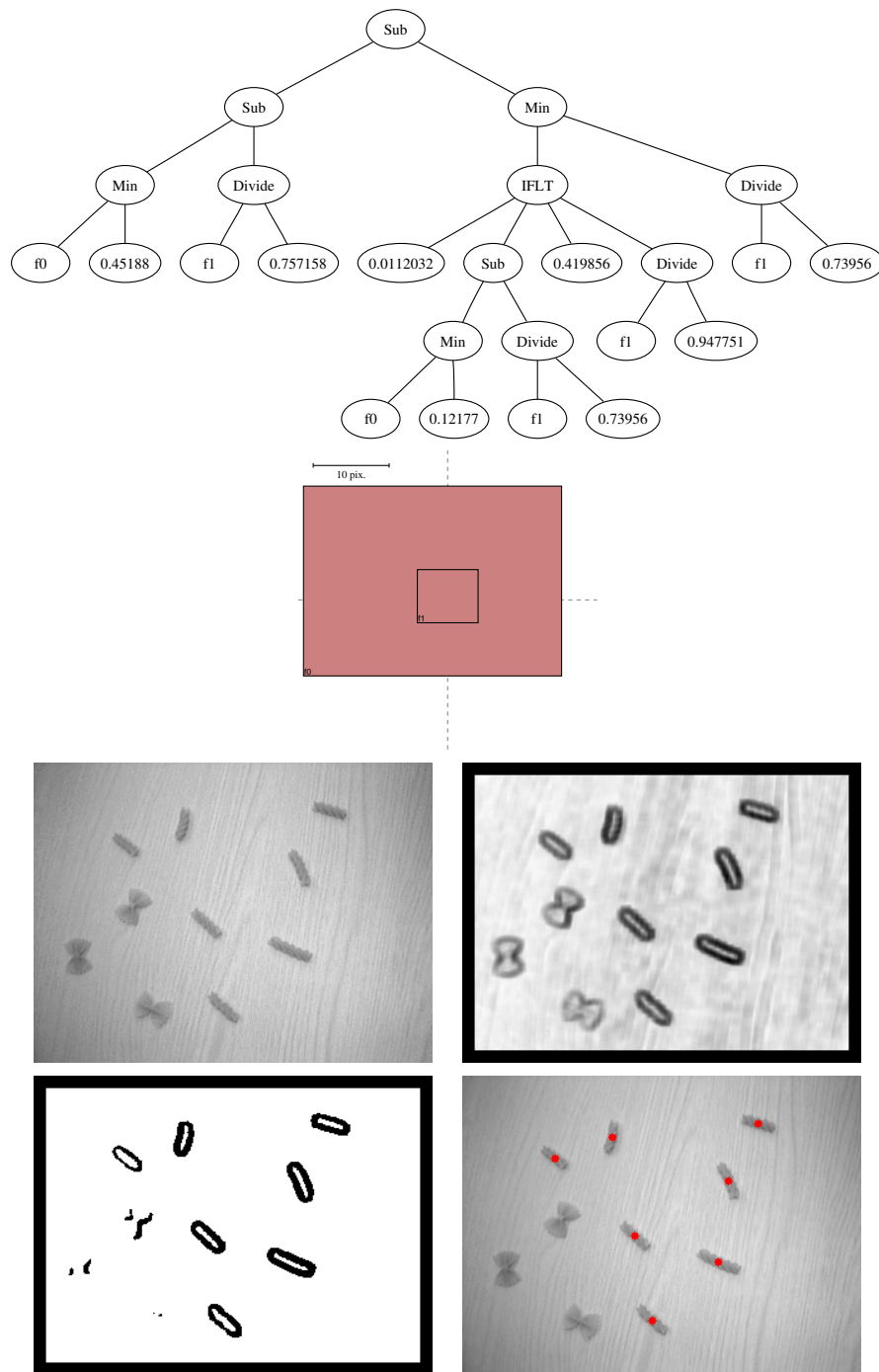


Figure 5.21: Best solution from 20 runs and sample results for set-P. The tree and features used are shown (top) along with sample output. Top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses.

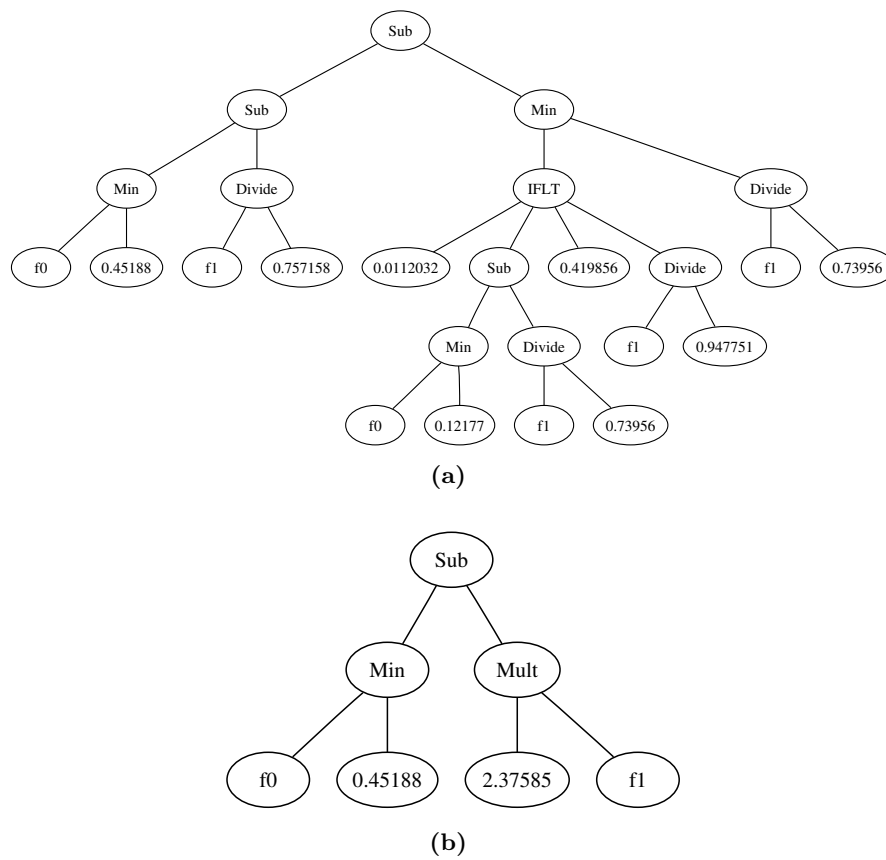


Figure 5.22: (a) The original solution for Set-P (b) the functionally equivalent simplified version (bottom)

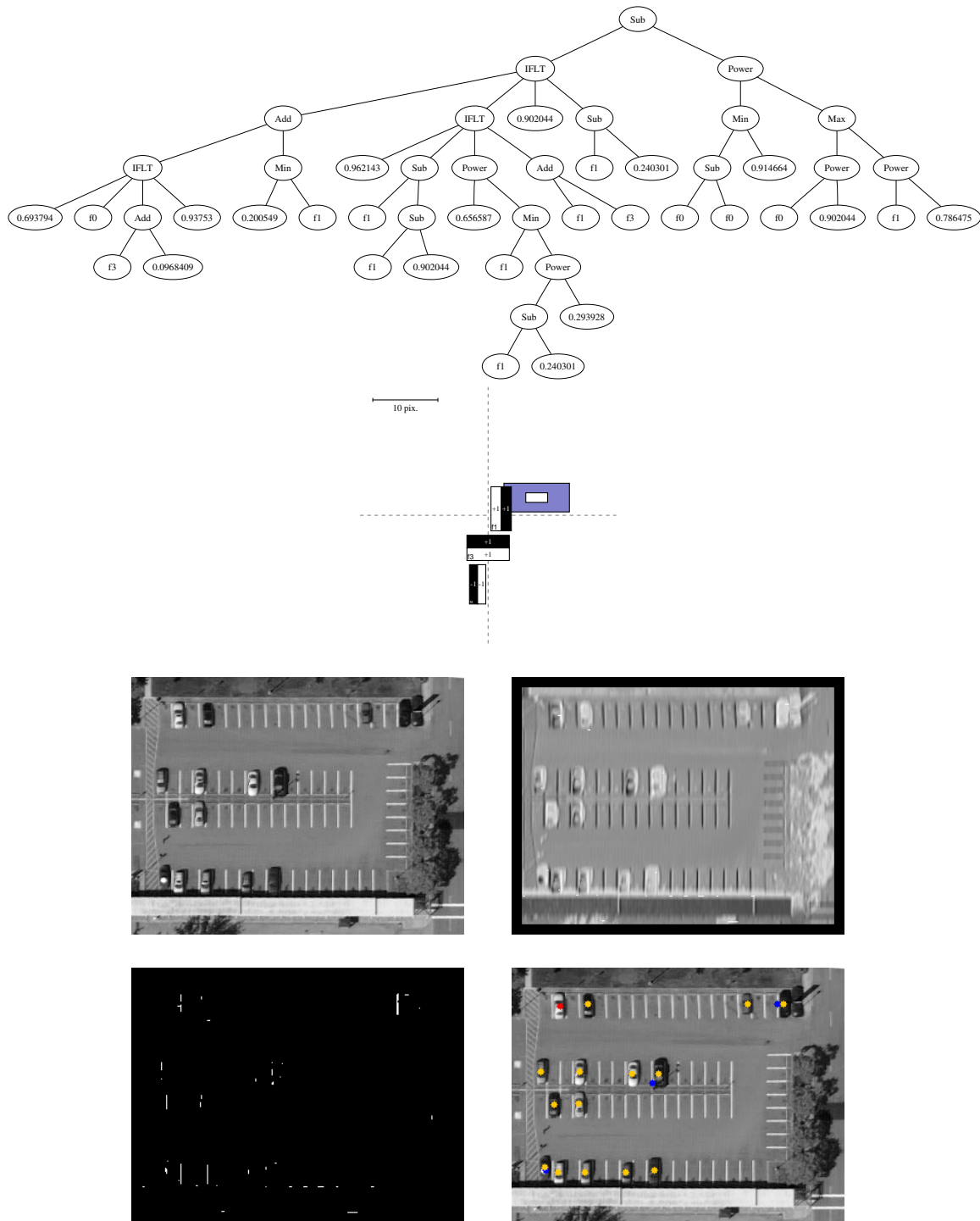


Figure 5.23: Best solution from 20 runs and sample results for set-AT. The tree and features used are shown (top) along with sample output. Top-left: the original image, Top-right: the raw detector output, Bottom-left: thresholded output, Bottom-right: final colour coded guesses.

The performance of the system was very good in several respects. Firstly the accuracy achieved in the individual tasks was very high, as can be seen from Table 5.2. The FOM figures in the region greater than 0.95 represent solutions with just one or two false positives or negatives over the entire datasets. This level of accuracy is likely to be high enough for most real-world applications. Secondly, the size of the solutions is worth noting. The average sizes of the trees was very small compared to other published solutions for similar problems. For example, in [124], solutions for similar tasks used in the order of 80 nodes. The number of features needed was also significantly smaller. The mean number of features used by the system was between less than 5 for all datasets, whereas published results use many times more as described in Section 3.1. This compactness and efficiency is a result of the fact that co-evolution allows the detector and features to complement each other. Whereas static feature systems might use up many tree nodes in order to compensate for inadequate information coming from the features, the co-evolutionary system can just change the features! This means we need fewer features in total and require less computation in the detector to deal with them.

5.8 Comparative Analysis

It is important to consider whether the co-evolutionary approach is significantly better than a traditional fixed feature set method. In order to assess this, a set of experiments was performed using a generic fixed feature set consisting of 20 features. This set is the one proposed by [124] and is detailed in Figure 2.7. All other settings remained the same – the only difference was that the feature set was pre-defined and not evolving. Twenty runs were performed on each dataset and the best test-set FOMs were recorded. Due to the nature of the data the Wilcoxon/Mann-Whitney⁴ test was chosen and was performed on the data to assess statistical significance. The results of this test can be seen in Table 5.3. In the cases of Set-T and Set-TN, there is not a statistically significant difference between the FOMs achieved by the two methods. However, it is important to remember that the fixed feature set contains 20 features, whereas the co-evolutionary method uses only a fraction of that number. So, in all cases, but especially those in which there is not

⁴These two tests are equivalent

Dataset	Fixed Features	Co-Evolving Features	Significance (p)
set-t	1.000 (0.000)	1.000 (0.000)	1.00 (0%)
set-tn	0.996 (0.012)	1.000 (0.000)	0.23 (76.9%)
set-tcn	0.930 (0.168)	0.997 (0.008)	0.02 (97.9%)
set-tcns	0.096 (0.047)	0.993 (0.020)	0.00 (100%)
set-tcnr	0.368 (0.031)	0.902 (0.057)	0.00 (100%)
set-tcnr	0.147 (0.046)	0.923 (0.054)	0.00 (100%)
set-kc	0.536 (0.082)	0.997 (0.008)	0.00 (100%)
set-p	0.170 (0.047)	0.945 (0.050)	0.00 (100%)
set-at	0.042 (0.026)	0.063 (0.027)	0.03 (97.5%)

Table 5.3: A comparison of the performance of fixed feature set experiments versus co-evolving feature set experiments. Wilcoxon/Mann-Whitney significance (two-tailed) is shown and for clarity the calculated confidence interval is shown in brackets. Values above 95% support the hypothesis that co-evolving features produces better performance than fixed feature sets. For those that are not significantly better in terms of FOM it should be remembered that they are significantly better in terms of feature set size – the fixed feature set contained 20 features, whereas the co-evolving set contained a maximum of 4.

significant gain in accuracy, it is interesting to note that the results were achieved with much fewer features.

It should be noted however, that the co-evolutionary approach used significantly more training time. This is due to the fact that a single evaluation actually contains a number of separate evaluations (the collaboration pool size). As a pool size of 10 was used in the experiments, it can be assumed that the co-evolutionary system used at least 10 times more training time than the fixed feature set approach. It would be interesting further work to consider what would happen if the fixed set approach was allowed to use 10 times more computation, by increasing the population size and/or the number of generations used. However, the improvement in fitness during fixed set runs tended to plateau after about 25 generations and it is doubtful whether increasing the training time would make any difference.

5.9 Limitations

The system found it impossible to find a good solution to the Set-AT problem, while apparently finding the simpler datasets very easy. The difference between these datasets is fairly clear - Set-AT is much more cluttered and the number of pixels occupied by targets is greatly outnumbered by the number of background pixels. Also, there are many different *types* of background pixels - roads, buildings, trees etc. The full-image method applies exactly the same operation at every pixel in the image, meaning that it is effectively trying to come up with one single global operation that can tell the difference between cars and all of these other types of object. This is not the way that effective problem solvers, such as humans, normally go about problem solving. They tend to break the problem up into subproblems, and not try to solve all of the problem in one go. If a computer vision expert was trying to solve this problem, he would probably use domain knowledge to specifically create algorithms which ruled out known non-targets (such as trees, shadows etc.) rather than trying to produce a single super-algorithm that did everything at once.

A side-effect of the full-image approach is the huge amount of computation required. To produce one single fitness score, a number of features must first be extracted from every pixel of every training image. For a dataset such as the Set-P, this would represent $50 \times 320 \times 240 = 3840000$ pixels. If four features were being used then this means that

we require $3840000 \times 4 = 1.536 \times 10^7$ feature extractions to get just *one single* fitness value. On top of this the GP tree must then be applied to each of the 3.84 million pixels. GP tree parsing is notoriously slow, and many GP applications would use a great deal less than this number of tree evaluations in an *entire* training run. Due to the use of the integral image, and by using highly optimised GP code, this entire operation can be completed on each image in approximately half a second using a 2.5GHz Pentium based PC, and on the entire training set of 50 images in approximately 25 seconds. However, this is of course the time taken for just one evaluation. This figure must be multiplied by the population size, the collaboration pool size, and the number of generations. For the Set-P problem this means that we require roughly $1500 \times 10 \times 40 \times 50 \times 0.5 = 15,000,000$ seconds, or approximately 173.6 days of CPU time per run. The number of multiplications used in those calculations is a good indicator of the scalability problems associated with this technique. If the Set-P data had a resolution of 640×480 then nearly 700 days of CPU time would be required, without even considering the fact that larger trees, more features, and larger populations may be necessary to deal with the more complex data. These vast run times are mitigated in this work by the use of large compute clusters, but obviously these types of resources are not widely available.

The lack of flexibility to deal with more complex data and the scalability issues are the two major problems with this technique. The fact that there are good results for many of the datasets is encouraging and shows that the underlying method is sound. The system *can* co-evolve feature extractors and detectors, and produce accurate and compact results. However, these qualities are not carried through into more complex datasets.

5.10 Summary

This chapter has presented a concrete implementation of the framework and concepts presented in Chapter 4. The experiments in this chapter used an approach where the features are extracted and detectors are applied at every single pixel in the training image, hence it is called the “full-image” approach. In order to find good solutions, a novel distance-based fitness function was introduced which minimises the distance between guesses and targets, and has a penalty for missed targets. Experiments were performed on a variety of datasets which tested various aspects of the system’s abilities such as rotational and scale

invariance. Some natural image datasets were also used to test the system’s performance in the presence of realistic noise, uneven illumination and image artifacts.

Although the system produced quite remarkable results (in terms of accuracy and solution size), it required a very large amount of computation time to train. Also, with larger, more complex problems the fact that the system is learning with the whole image at once meant that it became more difficult for the system to find a single solution that solved the whole problem simultaneously. The next chapter shows how the use of a staged method can allow a more flexible and computationally cheaper training process.

Chapter 6

Using the Framework – A Multistage Approach

In the previous chapter a method for using the co-evolutionary framework was introduced. This method, the *full-image* method, applied solutions to the whole image and generated fitness values from the output image that resulted. Although on the whole, the quality and size of the solutions was impressive, the system did not scale well to more complex problems. There were too many simultaneous goals in even a simple problem and the system could not take advantage of any sort of divide and conquer approach to problem solving.

In this chapter, it is shown how a sampling, multi-stage technique can be used within the co-evolutionary architecture. This technique provides faster evaluation and adds flexibility to the system allowing it to adapt to newer and larger problems more easily.

6.1 Principle of the Multi-Stage Method

The multi-stage approach to object detection was introduced by Howard et. al. [42, 43, 45, 46, 44]. It is based on the idea that instead of training a classifier using every point in an image, you can train a “quick” classifier on a random sample of the points, apply the resulting classifier to all of the points, and then train further classifiers to deal with the errors produced in the first stage. This produces considerable savings in computational requirements. Although Howard denotes these stages as *first-stages* and *second-stages*,

they are in this work referred to as the *primary-stages* and *specialist-stages*, for increased clarity.

The primary stage of this approach tries, usually without success, to solve the problem using only a sample of all of the pixels. The primary stage considers all pixels belonging to targets, and a random sample of non-target pixels. An object detector is then trained on this subset of pixels and at the end of the training the best solution is applied to every pixel in the images. As the solution trained only on a sample of pixels, this will of course generate a number of false positives, but in general, this number is very small relative to the size of the image. The primary stage therefore creates a basic detector that can hit most of the targets and rule out most of the non-target pixels. These remaining pixels are then passed on to specialist stages for further consideration.

In the specialist stages, classifiers are trained using the set of FPs and all of the target points produced in the primary stage, and have to discriminate between the two classes. These specialist stages have a harder task than the primary stage as their training set (the points produced by the primary stage) will all be very similar to each other. Training is performed in exactly the same way as the primary stage. Any point that is marked in the primary stage, and by any specialist stage, is considered as a “final guess”, and is used in the final fitness calculations. At the end of a specialist stage, a solution is produced and any targets that the solution hits are removed so that subsequent stages can concentrate on hitting the other targets. Several specialist stages successively add to the solution, hitting more targets but hopefully without adding too many false positives. The final output of the training is therefore a complete solution which contains one primary stage classifier, which is applied to every pixel, and a set of specialist stage classifiers which are applied to any pixel that the primary stage marked.

This multi-stage approach is very intuitive and mimics models of visual gaze and attention. In human vision it is known that the *gaze* very quickly finds objects of interest and directs the *attention* of the visual system towards them for more detailed analysis [37]. This phenomenon has been widely observed using eye-tracking techniques, and artificial systems (including an interesting evolutionary approach [24]) have attempted to replicate this behaviour. The multi-stage approach performs a very similar set of operations – the primary stage acts like a gaze system, identifying points that could be interesting and

deserve further consideration. Then, classifiers in the specialist stages analyse all of these points in more detail.

This chapter presents a method by which the co-evolutionary framework proposed in Chapter 4 can be used with the multi-stage approach, and how this pairing can be used to exploit the benefits of co-evolutionary feature extraction whilst improving on some of the shortcomings of the traditional full-image method.

6.2 Experimental Method

As with the previous approach, the dataset is first divided into training and test sets. For each of the training images, a set of training points is chosen as follows. The binary mask is used to extract all points that belong to the target objects. To this is added a random sample of the non-target points (the next section describes a more efficient, less random approach to this step). This is repeated for each training image, creating a large list of thousands of points and class labels which is used for training and fitness calculations. The number of target points in the list will be larger than the actual number of targets, N_t , as each target contains many pixels. The number of background points in the list is denoted as N_b .

In the primary stage of the system each individual in the population is evaluated using the entire point list, that is for each point we extract the features indicated by the solution, run the detector on the responses, and threshold the result. This thresholded value is compared with the actual ground truth and is classified as a true positive, false positive, true negative or false negative, and running totals of these figures are kept. Two more figures are also calculated – *hits* and *misses*. The *hits* count shows how many targets have been hit at least once. Although we want as many hits per target as possible, one hit on an object is all that is required for this hit count to be incremented. An opposite, and very harsh policy is enforced for false positive pixels. Every single pixel that is incorrectly marked increases the false positive count by one, regardless of whether they form contiguous areas or not. The *misses* value represents the number of targets that have

not been hit at all, and is calculated as $N_t - hits$. The fitness function below is applied to the counts that are returned.

$$f = \left(\left(1 + \alpha \frac{misses}{N_t} \right) \left(1 + \beta \frac{FP}{N_b} \right) \right) - 1 \quad (6.1)$$

The fitness function has two terms. The first term is the proportion of all possible targets that were missed and is weighted by the α parameter. The second term represents the proportion of all possible non-target points that were incorrectly marked as targets, and is weighted by the β parameter. Both terms should be minimised simultaneously and to encourage this they are multiplied rather than added, meaning that both must be low in order to minimise the whole expression.

In this primary stage we ideally want to hit all of the targets, even if this means that many false positives are generated. To achieve this, the α parameter is set several times higher than β . At the end of this primary stage the evolution will have produced a solution that hopefully hits all targets at least once and probably generates a number of false positives.

It should be noted that a *neutral zone* is established around each target. It is often impossible to mark the outline of a target exactly and errors are bound to occur. It would be very confusing for a learning system to encounter the conflicting information these errors would bring. Therefore, a 3-5 pixel thick neutral zone is created around each target (using dilation) depending on the size of the target and the estimated error involved. In this zone a guess is neither rewarded nor penalised.

The next step in the algorithm is to apply the evolved solution to the entire image. As the solution was evolved on only a sample of the image's pixels, it will obviously produce many more false positives when applied to the whole image. However, these new false positives are much fewer in number than the number of pixels in the image. The specialist stage of the algorithm evolves classifiers that can distinguish between the points returned by the primary stage.

The procedure for training the specialist stage solutions is exactly the same as for the primary stage, except that the training points being used are the set of all points returned by the primary stage, rather than the original set of target points and random non-target points. If any point receives a positive response in a specialist stage solution then it is

classified as a *final* guess, i.e. it is taken to be the final output of the algorithm. The fitness function used in the specialist stage solutions is the same as for the first, except this time the β parameter is higher than the α parameter, as we ideally want solutions with as few false positives as possible. After the specified number of generations, or if another termination criteria has been met (such as perfect fitness), the stage is over. At the end of the stage any target that has been hit with a final guess has all of its points removed from the list, even those that were not hit, as they would only serve to confuse the next stages. One of the most useful qualities of this technique is that the second stages specialise in different aspects of the problem. This is why it is so important for points or targets that have already been dealt with to be removed from the list. When the point list has been updated, the next specialist stage begins, in exactly the same way. The specialist stages are designated S_i, S_{ii}, S_{iii} etc.

At the end of the process, we have complete solutions which consist of a single primary stage solution which is applied to every pixel of an image, and a number of specialist stage solutions which are applied to the positive returns of the primary stage solution. Each solution consists, as before, of one detector and a user-specified number of feature extraction zones. This means that a complete solution is bigger than with the full-image approach, but this fact is outweighed by other factors, to be discussed later.

6.3 Non-Target Point Sample Selection

The amount of computation required for this technique is reduced by using only a relatively small sample of the non-target points. In object detection tasks it is often the case that the targets account only for a relatively small percentage of the total image area. It is also the case that the remaining non-target (background) points often have large areas of self-similarity i.e. there are lots of regions that look like other regions. Given this fact, it seems wasteful to apply an expensive learning process to every single pixel when a small sample could adequately represent the appearance of so many other pixels. The multi-stage approach exploits this fact by selecting a small sample of non-target pixels to use in training with the target pixels.

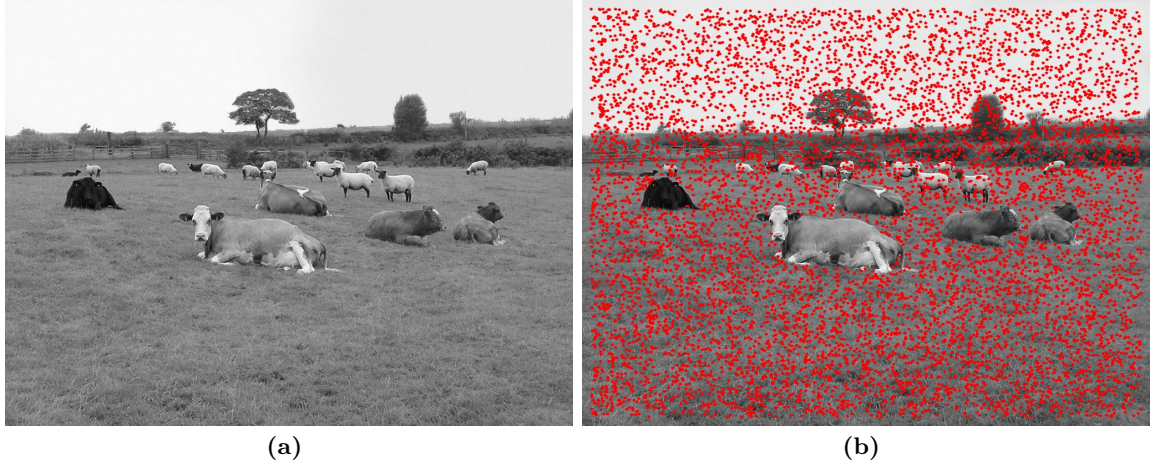


Figure 6.1: Uniform selection of points for a fictional cow-detection task. The uniform selection wastefully over-samples relatively uniform areas whilst under representing smaller, more confusing areas such as the sheep

6.3.1 Problems With Uniform Selection

In the work of Howard et. al., a number of points are randomly selected uniformly from the set of all background points. In their work ([46] for example) around 1000 points were taken from a 1024x1024 image. This ratio of about 1 percent will have a proportional effect on the amount of computation required for training. Ideally the background point sample size would be as low as possible whilst still providing a representative sample of the pixels in the image to allow effective training.

There is however a problem with a naïve uniform selection of points. This problem is really more of an inefficiency and is caused by the fact mentioned earlier that there are often large areas of self-similarity in the background points.

Consider the image shown in Figure 6.1(a) and the task of evolving a detector that can find cows (and only cows). The image is 1024x768 pixels and using the 1 percent rule, 7864 background points would be used for training. Figure 6.1(b) shows 7864 points uniformly selected from the non-target region.

Even though the sheep represent only one percent of the image area, they are actually the most troublesome distracting element in the image and deserve far more than one percent of the detector’s attention i.e. more points should be chosen on the sheep. However,

this is not the case with uniform sampling. As well as under-sampling this crucial class of pixels, the uniform approach over-samples some areas. In the example, the grass, which represents 55% of the total image area and the sky which represents 29%, do not need large numbers of samples as they are relatively uniform. These large uniform areas could easily be learned from using a much smaller number of points.

6.3.2 A Non-Uniform Method

The ideal situation would be one in which all parts of the image are fully represented but using the smallest possible number of points. It is impossible to pick a truly representative sample of points [46], but that does not mean that no effort should be made towards this goal. This section illustrates a simple method for non-uniformly selecting background points such that more emphasis is given to more “interesting” regions and less emphasis is given to uniform regions.

The approach taken is quite intuitive and follows from how humans would allocate a limited number of points. If we imagine an image consisting of an area of sky and an area of ocean, and we have to pick just two points, we would obviously divide them between the two classes and choose one sky point and one ocean point. If we could use 100 points, then it would make sense to choose 50 sky points and 50 ocean points. If there was also an area of beach shown, then we would probably put one third of the available points on the beach, one third on the sky and the final third on the ocean. In general we first divide the pixels in the image into a number of general classes, and then distribute the points equally within the classes. A few simple techniques allow us to create a similar effect automatically.

The first step of the algorithm is to try and allocate each pixel to a particular class of pixels. In order to determine the class, we calculate a number of properties of the pixel and the local area around it in much the same way as the feature calculations used later in the system. The properties are inspired by feature detectors known to exist (see [47] for details) in the human brain. The properties calculated are as follows –

1. Pixel mean around the point
2. Pixel variance around the point

3. Direction of the maximum edge response at the point
4. Magnitude of the maximum edge response at the point
5. Direction of the maximum bar response at the point
6. Magnitude of the maximum bar response at the point
7. Small centre-surround response at the point (7x7 window with 5x5 negatively weighted cutout)
8. Large centre-surround response at the point (13x13 window with 9x9 negatively weighted cutout)

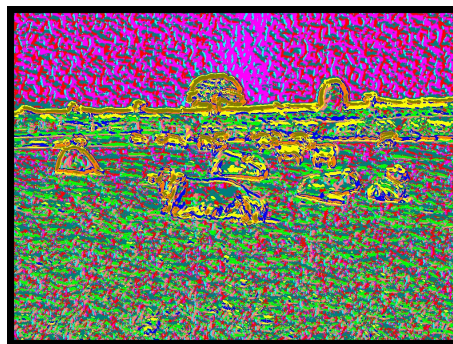
We use this eight-dimensional vector as the input to a k-means clustering algorithm which divides all of the pixels in the image into a number of classes (a user-specified parameter, 10 in this case) based on the responses. Once each pixel is assigned a class, an equal number of pixels can be randomly sampled from *each class*, meaning that we have equal amounts of each type of image area.

Figure 6.2 shows an example of this approach. Again, 7864 points are chosen. Firstly the k-means algorithm is used to classify each pixel, assigning it to one of 10 distinct classes. The result of this can be seen in the second image, where each colour represents a different class. Finally, one point is taken at random from each class until the target number of 7864 is reached. It can be seen that the distribution of the points is better than the uniform selection, especially with regard to the number of points chosen on the sheep which were previously under-represented. In fact, the distribution of the points is so much better, that we could use fewer points and still have a good level of representation of each class. The last figure shows the same image but this time with only 3932 points used (0.5% of the image size). Using this technique, the smaller amount of points still represents the different image areas thoroughly and for example still contains more sheep pixels than the original uniform selection. The actual number of points selected, the background sample size (BSS), is still a user specified parameter. It is possible that this non-uniform technique could be modified to choose its own BSS, but this is beyond the scope of the current work.

This non-uniform approach to non-target point selection brings two key advantages. The first and most important is that the points are specifically chosen to be ones which



(a) Original Image



(b) Pixel Class Image



(c) Final Points



(d) Smaller Final Point Set

Figure 6.2: Non-uniform selection of non-target points. The second image shows the result of the k-means clustering, and the two bottom images show the selection of 7864 and 3932 of these points respectively.

represent the major types of image region present, thus enabling much more effective training to take place. The side-effect of this procedure brings the second advantage – as the points are chosen more intelligently, we can use less of them. In a system which is heavily bound by raw computational power, this is a great advantage.

6.4 Basic Object Detection Experiments

The same datasets from the previous chapter were used to experimentally test the abilities of this system. As in the previous chapter, the *best* results achieved on each of the datasets are described in the following section. For each of these, the solution and an example of the output is shown. The colour scheme of the output points, repeated here for convenience is as follows – true positive points are shown in red, false positives are shown in blue and missed targets are shown in yellow. Any guesses in the neutral zone (as described in Section 6.2) are shown in green.

The function and terminal sets are exactly the same as those used in the previous chapter for the full image approach. In fact, the entire process is very similar apart from the core of the fitness calculation. The parameters used in the multi-stage experiments are summarised in Table 6.1.

6.4.1 Experiments on Set-T

The set-T problem is too easy for this system. A solution was found in generation zero (the random generation) in which the detector had one node returning the value of one feature. The feature calculated the mean of a small area around the pixel of interest. This resulted in an FOM of 1.0 on all training and testing images. No specialist stages were necessary. This primary stage solution used a background point sample size of just 0.2%, which on this dataset equates to 180 pixels.

6.4.2 Experiments on Set-TN

The addition of noise made this task harder than Set-T. However, after just 4 generations of the primary stage a solution was produced which hit all targets in all training images. When applied to the entire images, this solution produced no false positives i.e. it completely solved the problem using only the primary stage. When applied to the unseen

Dataset	T	TC	TCN	TCNR	TCNS	TCNSR	KC	P	AT
BSS (%)	0.2					0.4			0.7
specialist stages	3								4
α (primary stage)	3								
β (primary stage)	1								
α (specialist stage)	1								
β (specialist stage)	7								
μ_d	1500								
μ_f	100								
$cross_d$	0.7								
mut_d	0.2								
$cross_f$	0.35								
mut_f	0.65								
G (<i>per stage</i>)	40								
p_{tarp}	3								
$tourn_d$	6								
$tourn_f$	4								
d_{create}	6								
d_{mut}	3								
runs	20								

Table 6.1: Parameters used on the multi-stage experiments. Most parameters are common to all experiments, except the background sample size (BSS) and the maximum number of specialist stages used.

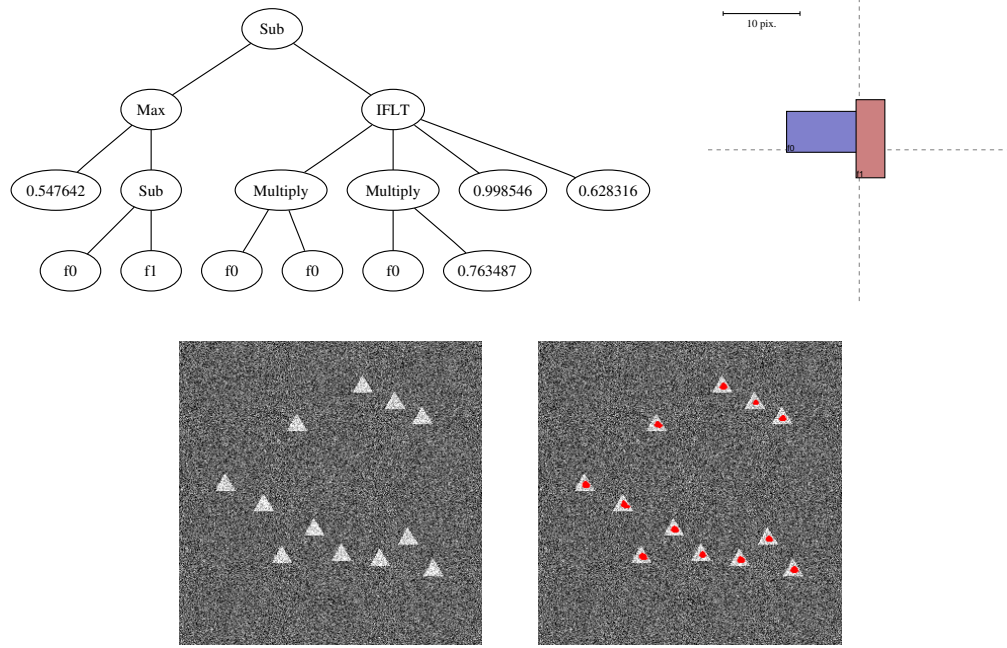


Figure 6.3: The primary stage solution (top) and sample outputs (bottom) for a solution to the set-TN problem. No specialist stages were needed for this problem as this solution classified all images correctly in the primary stage.

test set, the solution scored a perfect FOM of 1.0. This solution and an example of its behaviour can be seen in Figure 6.3. In this experiment only 0.2% of the non-target pixels were used in the primary stage training, which equates to 180 pixels per image. Training using only this small fraction of all of the pixels meant that the actual training time was less than one minute. The full image approach to this problem took several hours to train.

6.4.3 Experiments on Set-TCN

Even though Set-TCN is considerably more difficult to solve than Set-TN, the system did not require a significantly more complex solution. One hundred and eighty background points were used from each image. Within three generations, a primary stage solution was found which achieved perfect fitness and FOMs on the sampled points. When applied to the whole image, a small number of false positives were found on some of the circle elements. A single specialist stage was needed to remove all of these false positives and

produce a perfect FOM of 1.0 on all training and testing images. The solution produced was remarkably simple, using just three nodes and two zones in both the primary stage solution and the specialist stage solution. This solution is shown in Figure 6.4. The solution is actually more concise than the solution evolved for set-TN, and so it could easily be used for that set also.

6.4.4 Experiments on Set-TCNR

The primary stage training of the Set-TCNR solution did not terminate early and ran for the full 40 generations. This was due to several false positives in the sampled point set. However, the solution was still of a very high quality, and when applied to the whole images produced an average of only 289 false positives per image. Two further specialist stages were required to deal with these false positives. After these stages a perfect solution with FOM of 1.0 was produced on the training images. When applied to the unseen test set the average FOM was 0.996. The solutions are shown in Figure 6.5. It can be seen from that the primary stage solution managed to rule out the majority of the background pixels, but incorrectly hit all of the circles. This precisely illustrates the purpose of the multistage approach – the primary stage rules out all of the ‘easy’ background pixels (which are normally the majority) and sends the ‘harder’ pixels onto the specialist stages for further consideration.

6.4.5 Experiments on Set-TCNS

Only four generations were required to evolve a primary stage solution which correctly classified all of the sample points (i.e. all target points and in this case 180 background points per image). This solution was remarkably simple, using just three nodes and two features. This simplicity is testament to the fact that the features can evolve, meaning that much simpler detectors can be produced than when the detectors have to compensate for features that do not capture the essence of the problem. When applied to the entire images, this solution produced *only four false positives in the entire set of 50 training images* i.e. the primary stage almost completely solved the problem in its own right. A single generation of stage S_i was needed to clear up these false positives, and produced an even simpler solution which just directly returned the value of a single feature. When

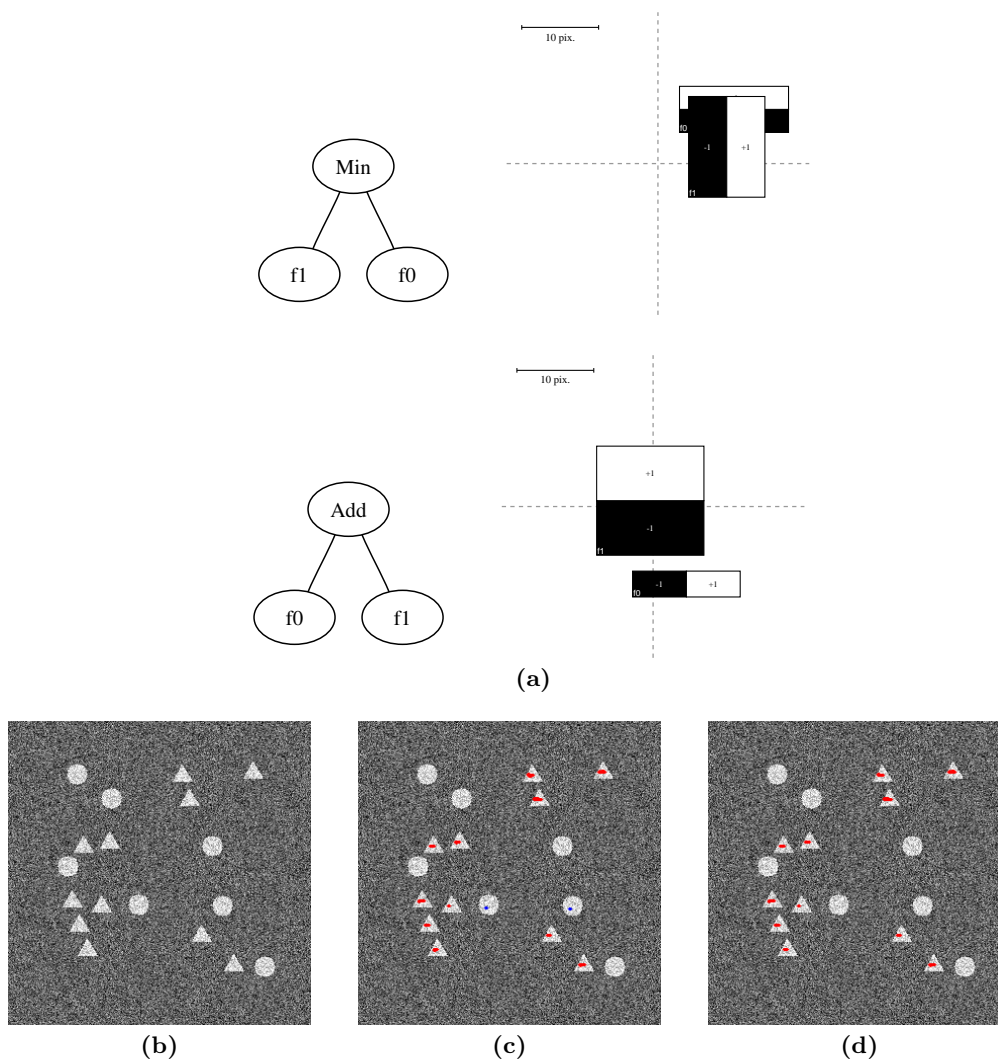
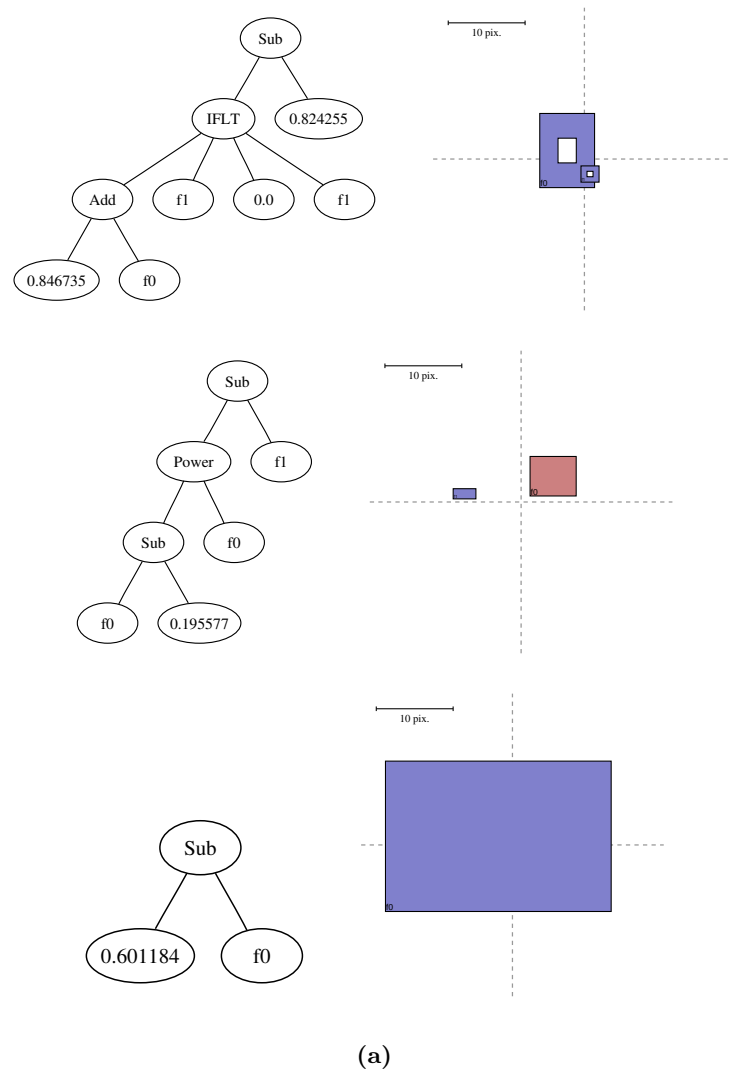
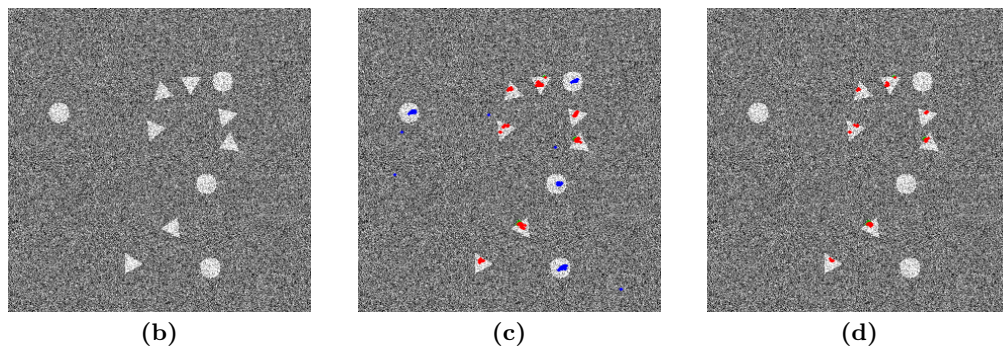


Figure 6.4: Best solution (from 20 runs) for the Set-TCN problem. (a) Shows the primary stage solution (top) which is very concise and has good performance, producing only a small number of false positives. The similarly concise specialist stage solution (bottom) deals with all of these false positives producing a perfect result. (b) Shows an unseen test image (c) Shows the primary stage output (d) Shows the output after the specialist stage



(a)



(b)

(c)

(d)

Figure 6.5: Best solution (from 20 runs) for the Set-TCNR problem. (a) Shows the primary stage solution (top) and then the two specialist stage solutions (b) Shows an unseen test image (c) Shows the primary stage output (d) Shows the output after the specialist stage

applied to the test set, this complete solution (shown in Figure 6.6) produced an average FOM of 0.9901. As the primary stage solution performed so well, it was tested in its own right on the unseen test set and produced an average FOM of 0.982. Although not as accurate as when combined with the stage S_i solution, this figure is still very impressive. It is even more impressive because of the conciseness of the solution. It is doubtful that any hand crafted solution could be quite so efficient. The configuration of the primary stage features is finely tuned to the task of triangle detection. The horizontal feature would lay across the thin part of the triangle and therefore also over some of the background. This would produce a high response over a triangle while producing a very low response over areas of background or over circles, where the approximately uniform colour would cancel the two sides of the detector out.

6.4.6 Experiments on Set-TCNSR

The best solution for Set-TCNSR is shown in Figure 6.7. In this solution the primary stage ran to completion (40 generations) and although it did not produce a perfect solution in terms of fitness, the solution it did find was still more than adequate for input to the specialist stages. When applied to the complete images this primary stage solution produced an average of 119 false positives per image. Three specialist stages were applied to the target points and these false positives. At the end of the specialist stages the complete solution had an average FOM of 0.958 (s.d. 0.06) over all of the training images and an average FOM of 0.876 (s.d. 0.10) over all unseen test images.

6.4.7 Experiments on Set-KC

The primary stage training on set-KC ran to completion without finding a perfect solution on the sampled points. When applied to the entire images this primary stage solution produced an average of 96.8 false positives per image. Three further specialist stages were necessary to produce a solution that hit all targets but had the side effect of producing some false positives. Stages S_i and S_{ii} ran to completion and stage S_{iii} terminated after three generations as all remaining targets were hit. In terms of the FOM on the training images, the solution scored an average of 0.948 (s.d. 0.08). When applied to the unseen

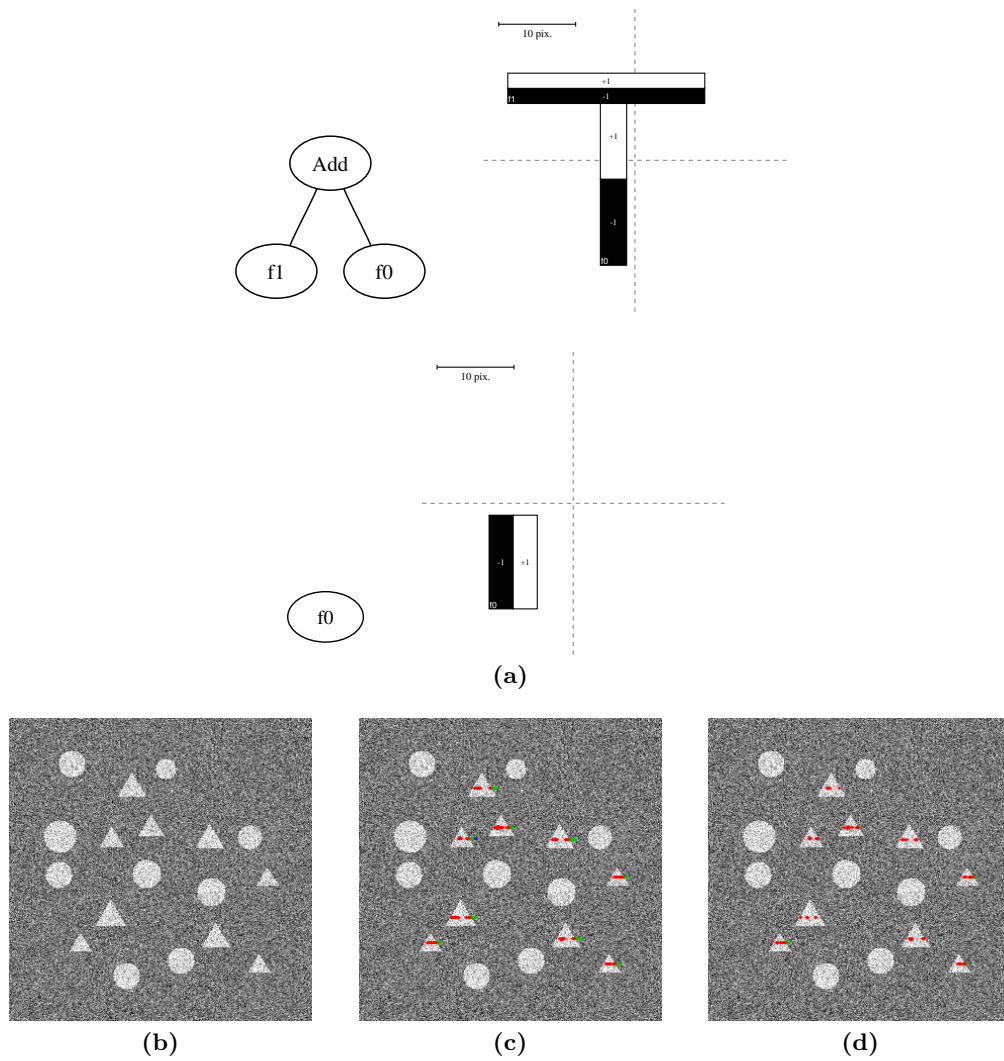


Figure 6.6: Best solution (from 20 runs) for the Set-TCNS problem. (a) Shows the primary stage solution (top) and then the two specialist stage solutions (b) Shows an unseen test image (c) Shows the primary stage output (d) Shows the output after the specialist stage

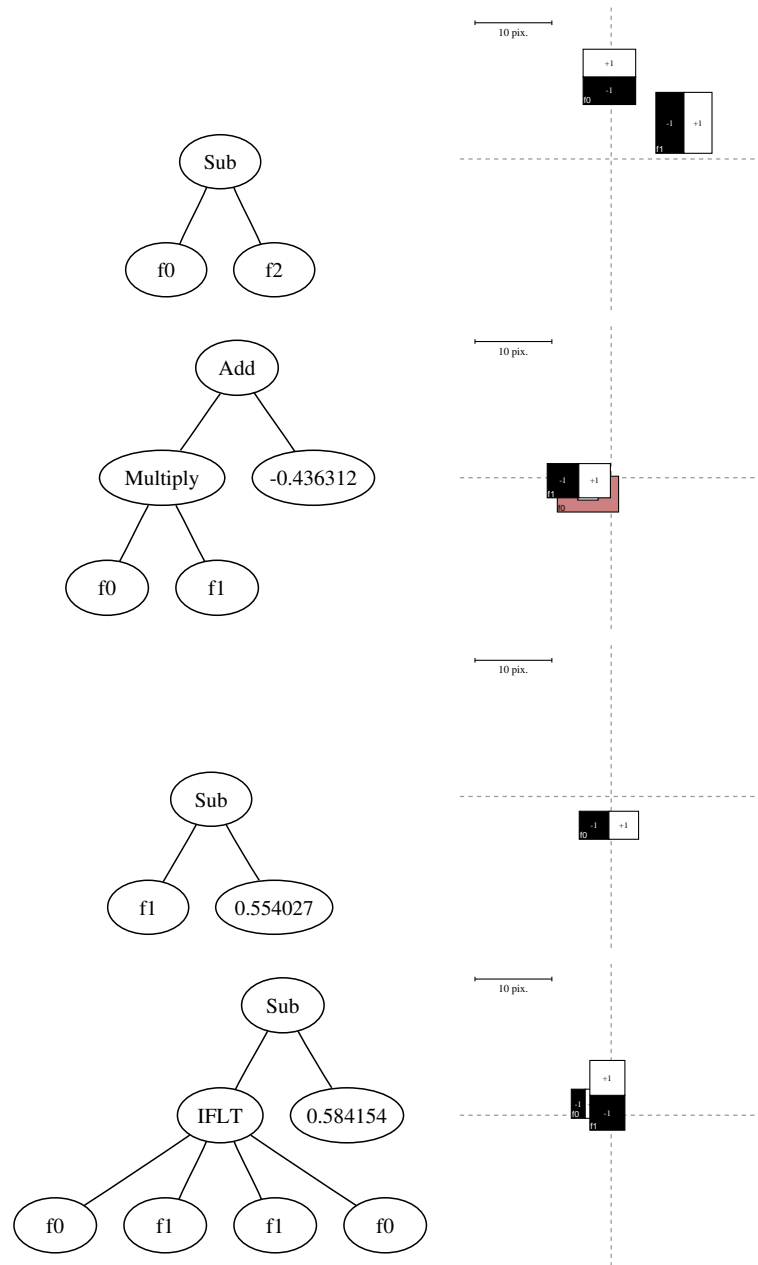


Figure 6.7: Best solution (from 20 runs) for the Set-TCNSR problem. Shows the primary stage and three specialist stages.

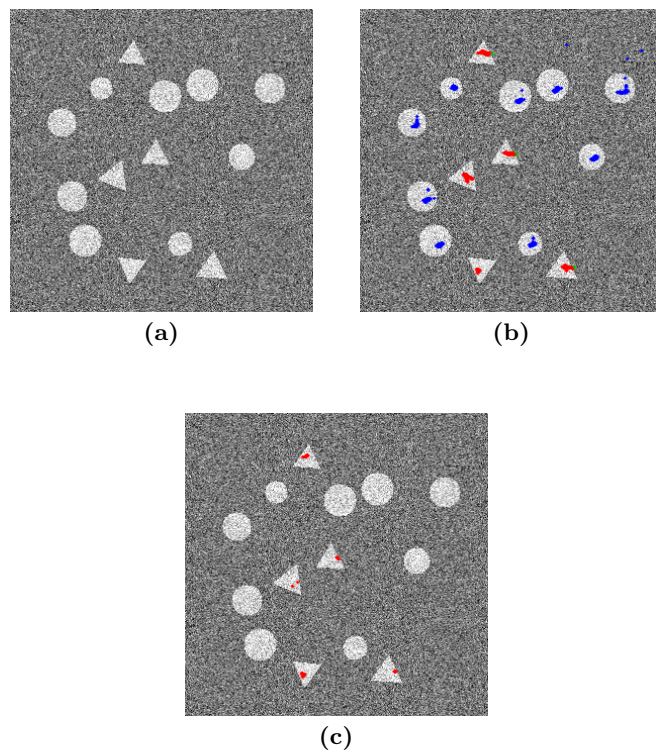


Figure 6.8: Sample output from the set-TCNSR solution shown in Figure 6.7. (a) Shows an unseen test image (b) Shows the primary stage output (c) Shows the output after the specialist stages.

testing images the average FOM was 0.917 (s.d. 0.11). The solution can be seen in Figure 6.9.

6.4.8 Experiments on Set-P

On the pasta detection task, a suitable primary stage solution was evolved after 24 generations which hit all targets and when applied to the whole image produced an average of 29 false positives per image. The specialist stages were trained on these false positives and the target points, and after all 3 specialist stages there were only 11 false positives over all of the 50 images and all of the 195 targets were hit. Overall this produced an average FOM of 0.95. When applied to the unseen test set of 50 images, the solution missed 15 of the 221 targets and generated 18 false positives producing a FOM of 0.86. This means only around 6% of all targets were missed, with a relatively small number of false positives. The best solution can be seen in Figure 6.11 and sample output can be seen in Figure 6.12

6.4.9 Experiments on Set-AT

The Set-AT dataset is by far the most difficult of all the problems. There are many different types of non-target pixels and a great deal of variability in the targets. For this reason the non-target sample size parameter was set to 0.9% for the experiments. The primary stage of the training produced a 69 node solution which used 4 features. This solution correctly classified all but 7 of the 10,107 sampled points. In spite of this good performance, when applied to the entire images, an average of 271 false positives were produced per image. This primary stage solution can be seen in Figure 6.13.

Four specialist stage solutions were trained on the points marked by the primary stage. The solutions evolved can be seen in Figures 6.14–6.17. After all of these specialist stages, the final FOM on the training images was 0.810, and the final FOM on testing images was 0.765. These figures are lower than in other experiments and reflect the difficulty of the task, but still represent an acceptable level of performance¹. Similarly, whilst the size of the solutions is greater than was seen for other datasets, they are still relatively small and

¹It should be remembered that unlike other accuracy measures where 0.5 represents random performance, the FOM does not, and lower values like this may still represent good performance

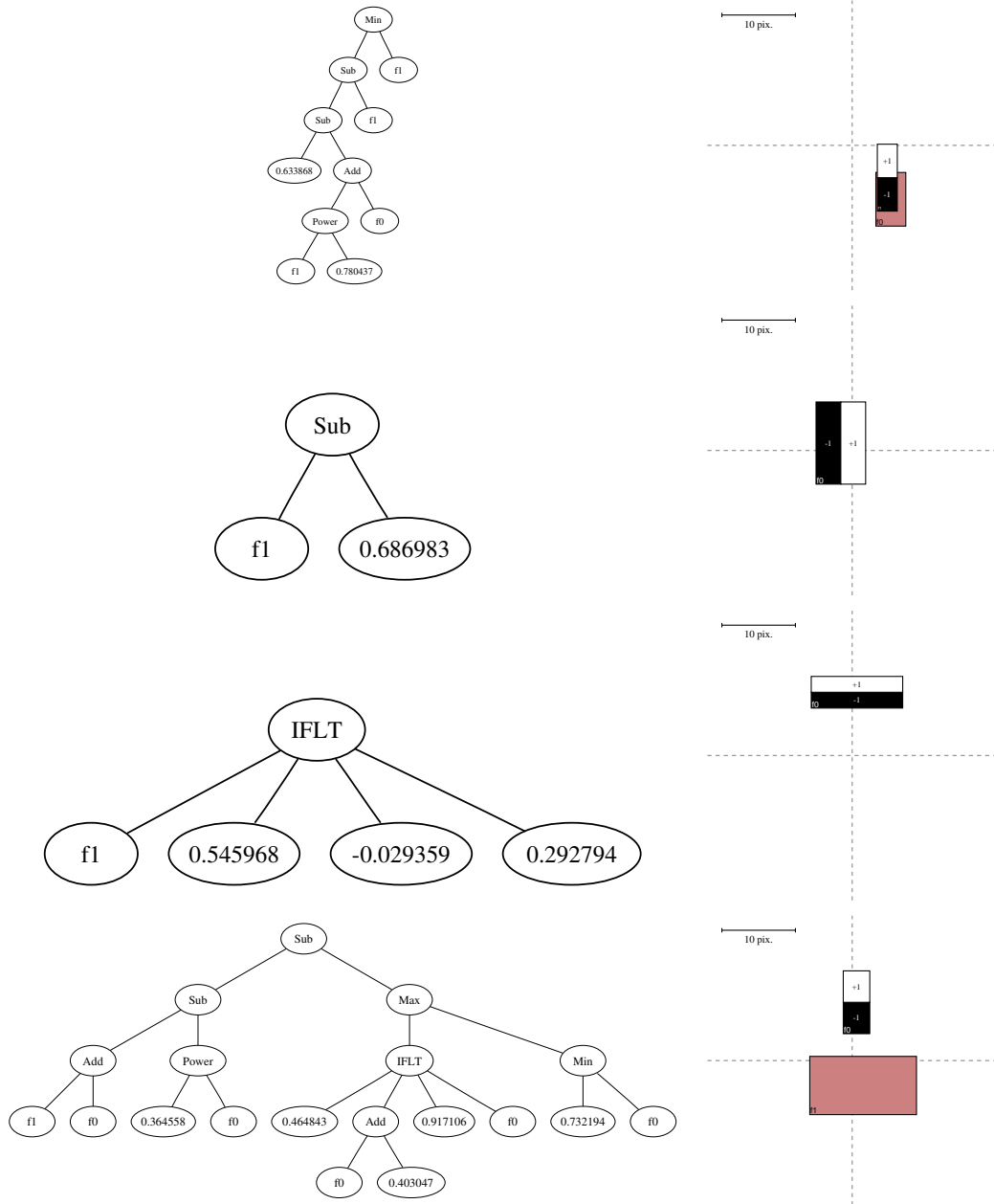


Figure 6.9: Best solution (from 20 runs) for the Set-KC problem.

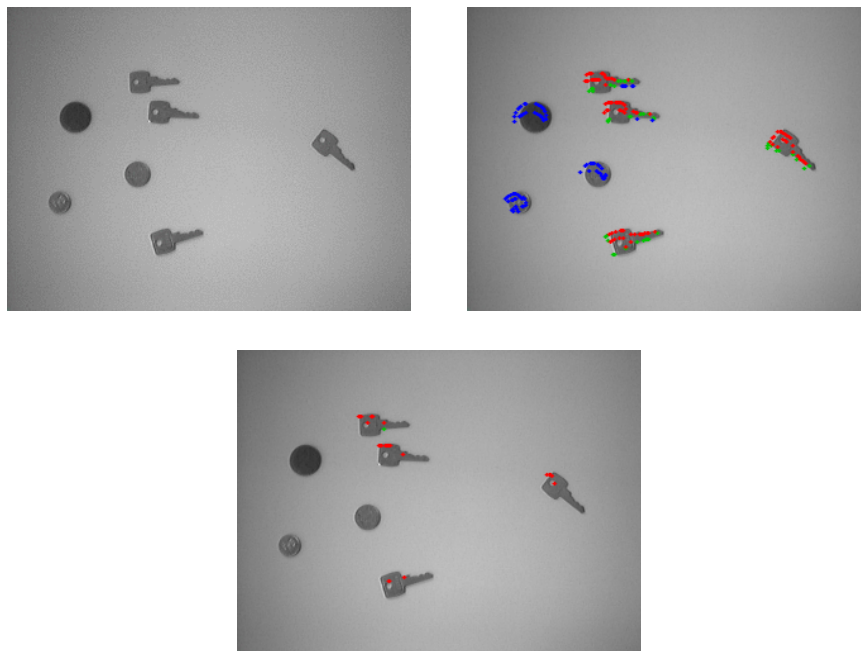


Figure 6.10: Sample output from the Set-KC solution shown in Figure 6.9. The figure shows the original image, the primary stage output and the final stage output.

when parsed take less than a second to execute. When compiled the response is almost instantaneous. Sample output from this solution can be seen in Figure 6.18.

6.5 Further Experiments

6.5.1 Solution Size and Performance

As with the previous chapter, the results shown in the previous section are the *best* results produced by the system. Table 6.2 shows the mean performance over 20 runs of each experiment. It should be noted that in this table, the size, number of features used, and number of generations used are summed over all of the stages. This table illustrates the comparative difficulty of the various tasks. For example all of the Set-AT experiments ran to completion (200 generations, split over 5 stages) i.e. not a single run managed to find a solution which hit every target.

One of the major advantages of the multistage approach is also one of its biggest disadvantages. As the problem is successively broken down into smaller sub-problems the approach is very flexible and can deal with a large intra-class variability (as there

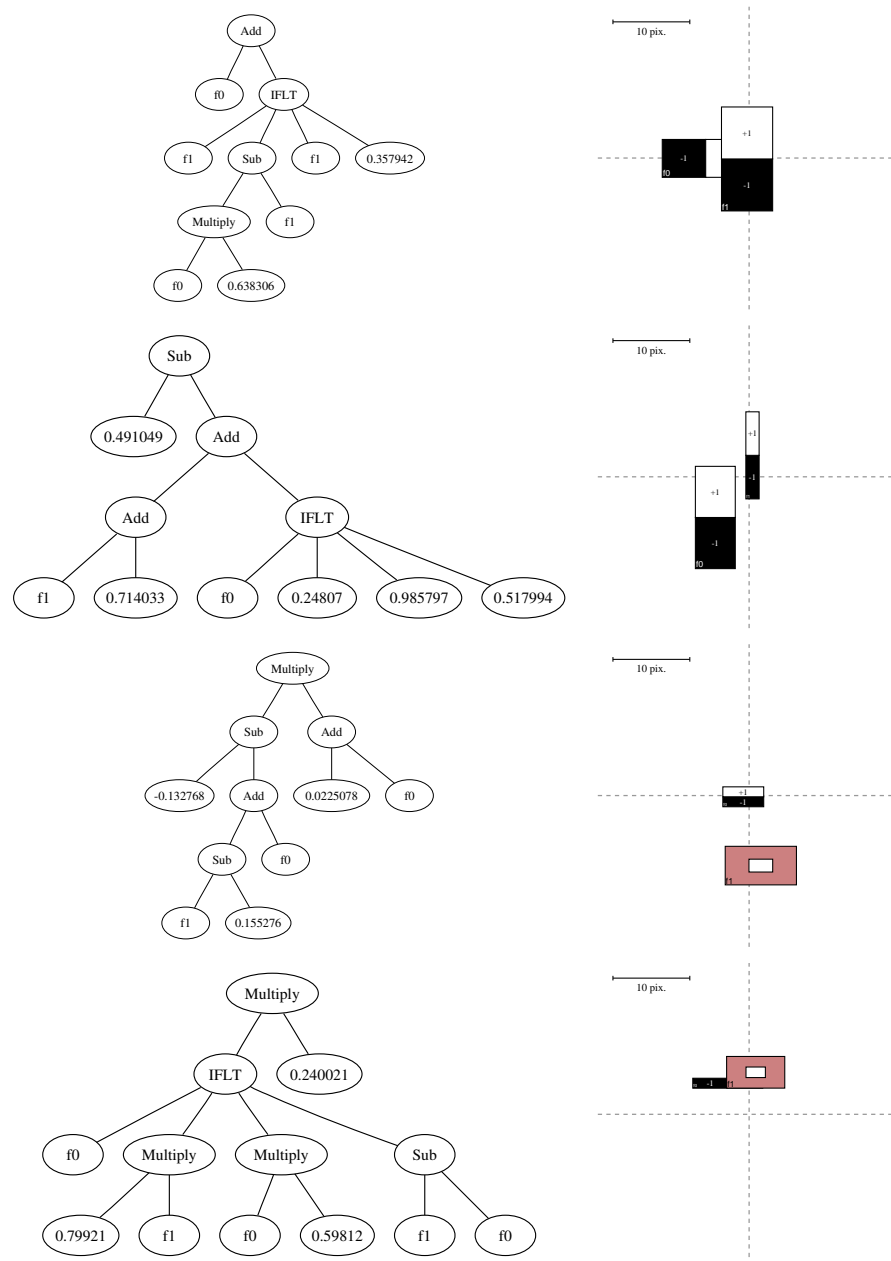


Figure 6.11: Best solution (from 20 runs) for the Set-P problem. This solution consists of one primary stage and three specialist stages. The figure shows the detector and feature extractors for each stage.

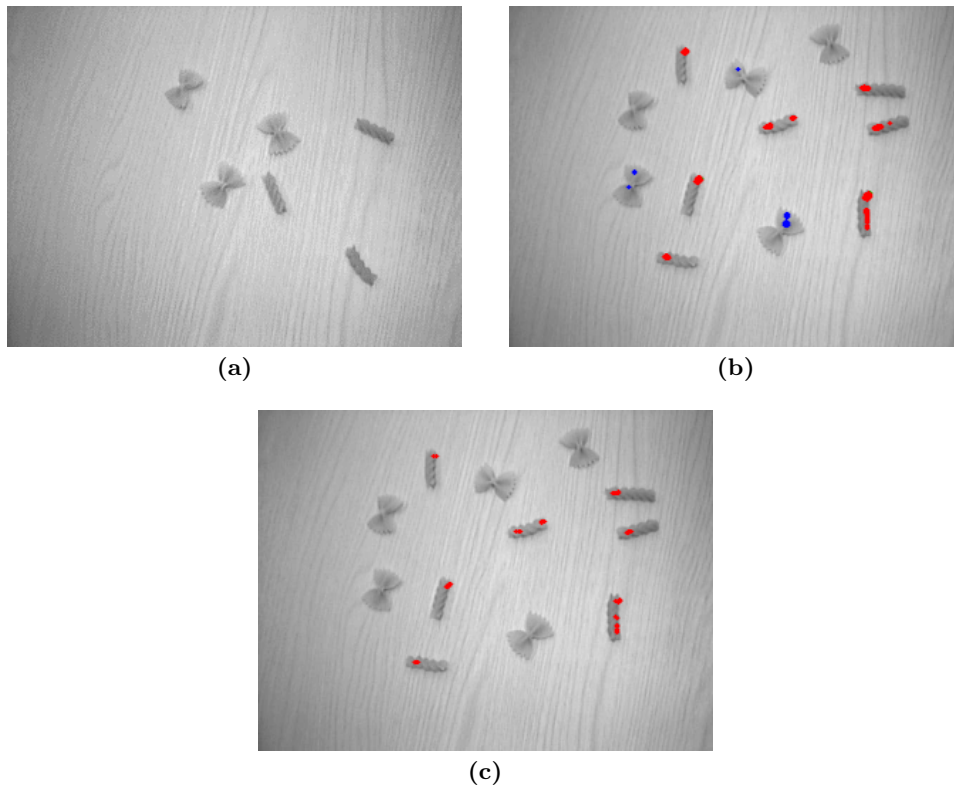


Figure 6.12: Example output from an evolved Set-P detector. (a) Shows the original image. (b) shows the points marked by the primary stage. The detector has learned to distinguish between the targets and the majority of the background. (c) show the points marked after the specialist stages are complete. The false positives are eliminated.

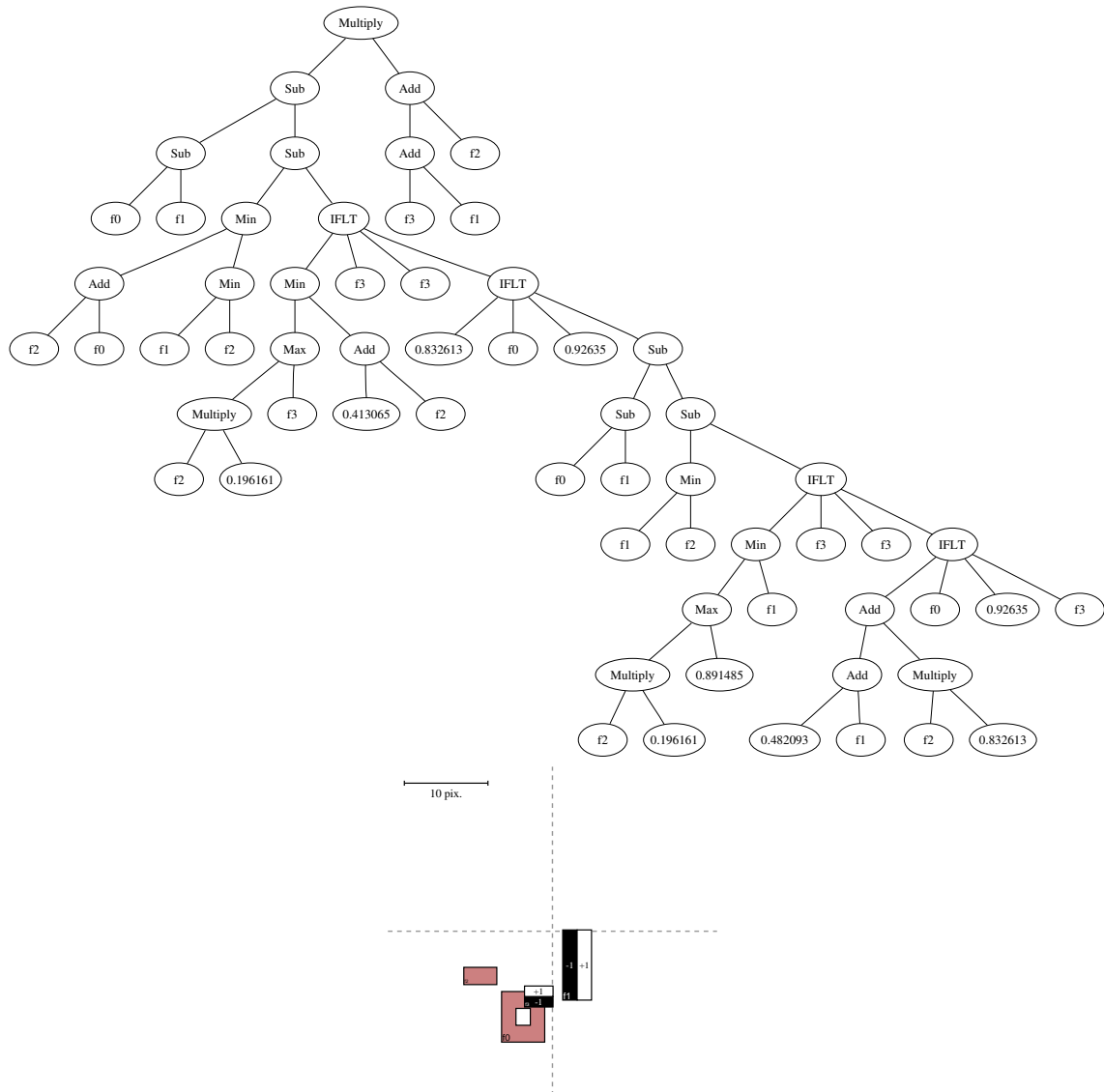


Figure 6.13: Primary stage of the best solution (from 20 runs) for Set-AT

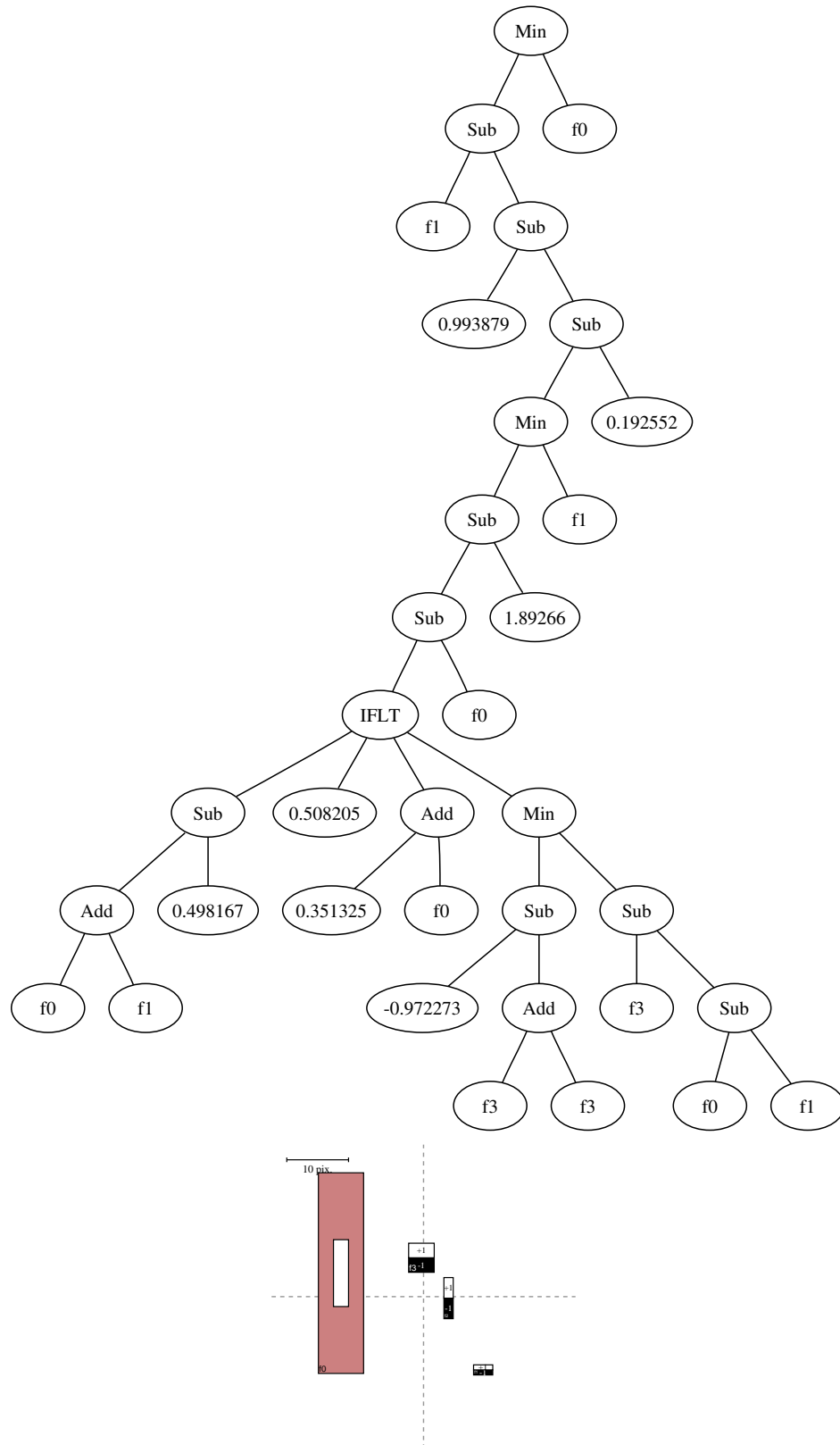
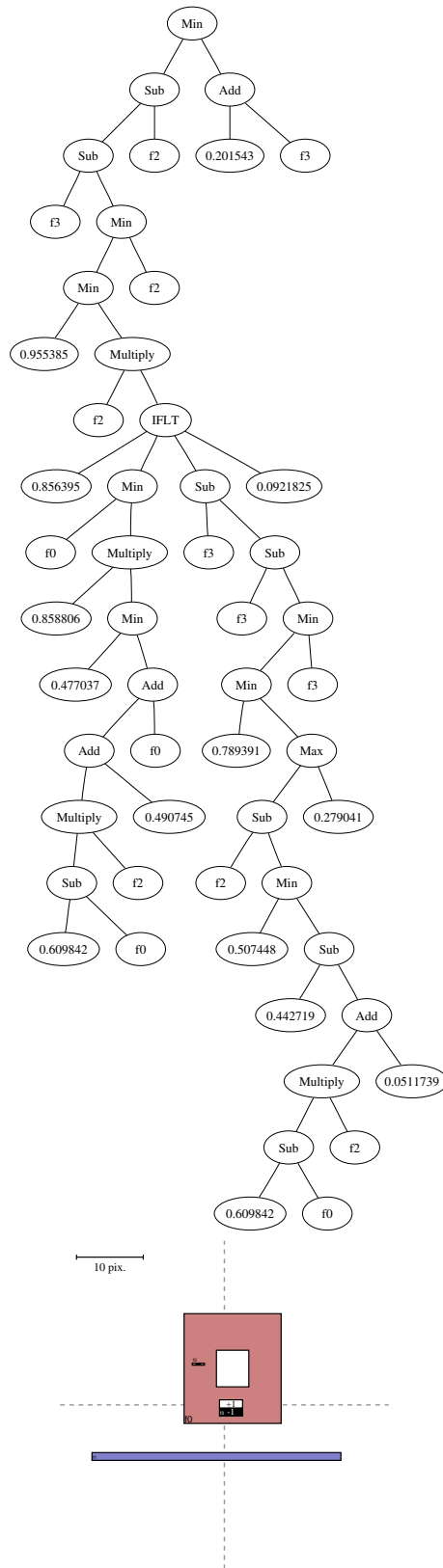


Figure 6.14: Stage S_i of the best solution (from 20 runs) for Set-AT

Figure 6.15: Stage S_{ii} of the best solution (from 20 runs) for Set-AT

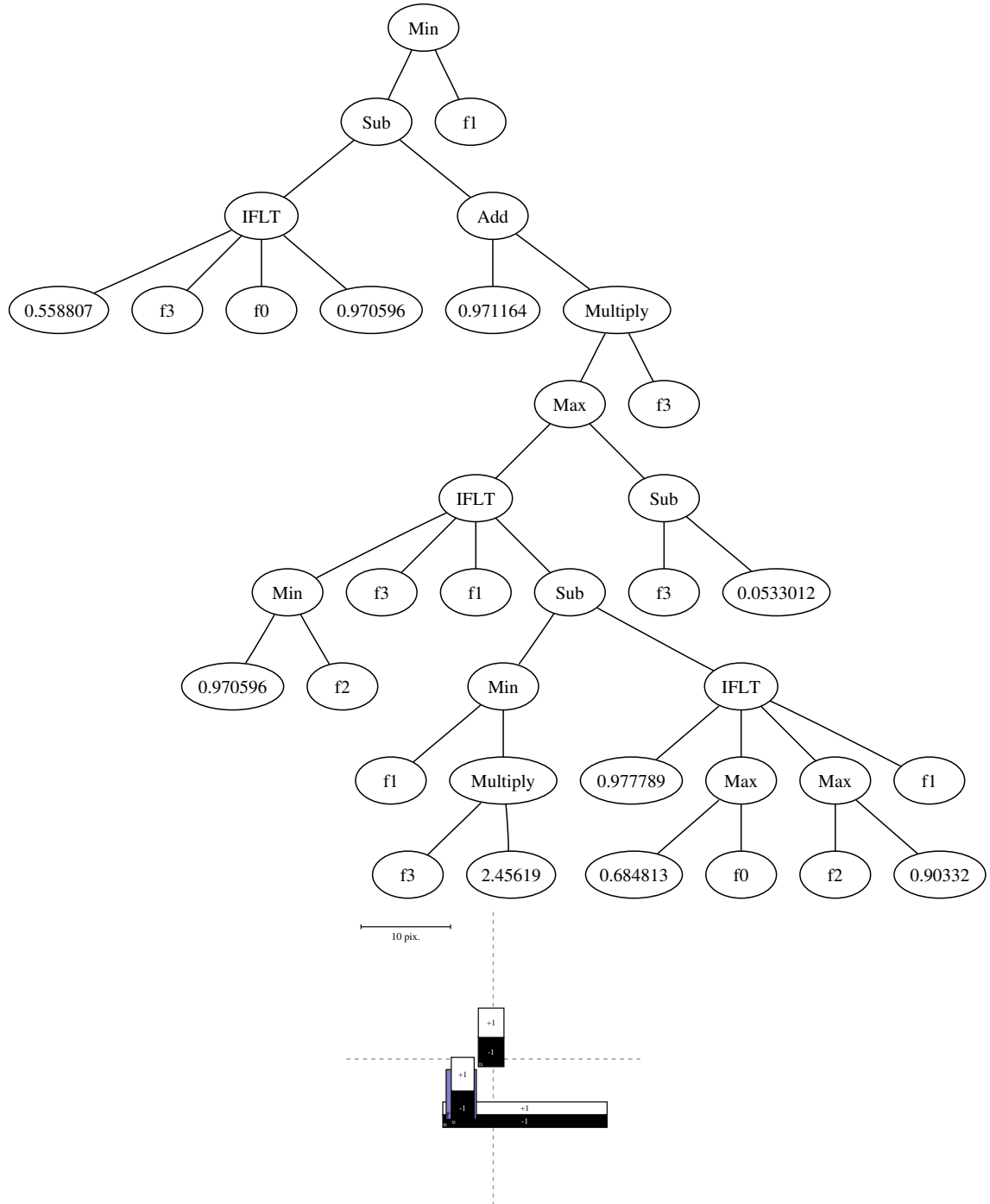


Figure 6.16: Stage S_{iii} of the best solution (from 20 runs) for Set-AT

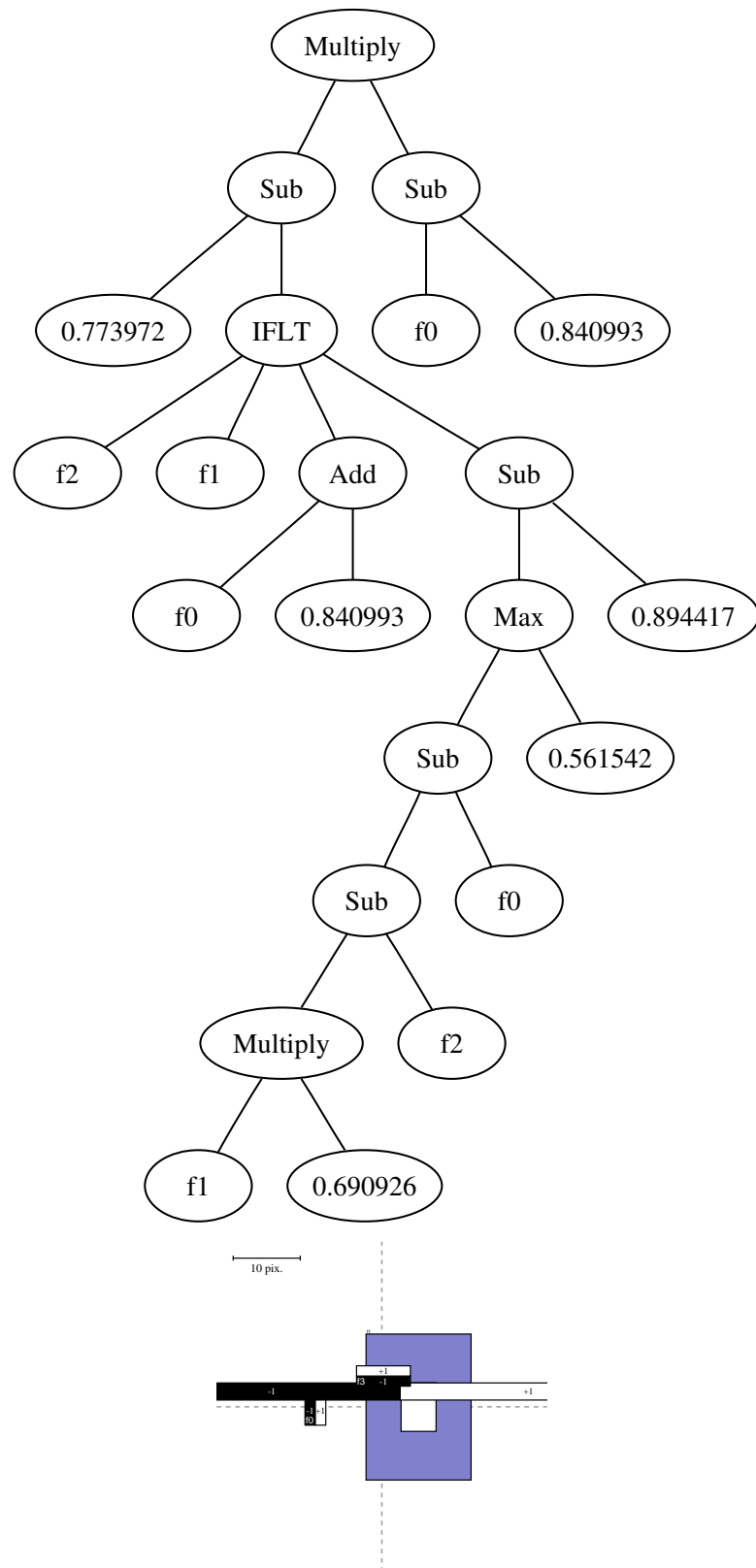


Figure 6.17: Stage S_{iv} of the best solution (from 20 runs) for Set-AT



Figure 6.18: Sample output from the Set-AT solution shown in Figures 6.13-6.17. The figure shows the original image, and the final stage output.

is in Set-AT with many shapes and colours of cars) and also deal with a wide variety of background types. However, this flexibility comes at a price – for each stage a new solution is generated meaning that the complete solution has the potential to be much larger than a single solution produced by the full-image approach. Also, as a new set of features is used for each stage, it is also likely that the complete solution will use more extracted features in total. The mean solution sizes for the full-image and multi-stage solutions are shown in Table 6.3. It can be seen that in most cases the number of features used is greater for the multi-stage approach. However, it is important to remember that this increase in size is coupled with an increase in the system’s ability to tackle more complex problems. In the Set-AT experiments the solutions used by the full image approach were small but these did not come close to solving the problem. The multi-stage approach to the same problem used many more nodes but also produced well-performing solutions². It was initially feared that the increased size of the multi-stage solutions would produce a proportional increase in the execution time of the solutions, however this is not the case. Although the multi-stage solutions contain more nodes and use more features, this will not necessarily produce a proportional increase in execution time for the following reasons.

²As Tarpeian bloat control was used, there was no upper limit on the size of the trees in both types of experiments, so the full-image approach did have the option of using a larger tree.

1. In the specialist stages, the solutions are applied to only a tiny fraction of the image’s pixels, rather than the whole image as is the case in the full-image method and in the primary stage.
2. Although a solution often has several specialist stages, they will not all be executed for all points. As soon as a specialist stage marks a point as a final guess, the point is removed. The number of points going on to subsequent stages decreases dramatically.

These two points mean that the impact of the specialist stages is almost negligible, and that the time to execute a multi-stage solution is only marginally larger than a similarly sized full-image solution. It is of course difficult to accurately compare the execution times of the solutions as they have different performances. For example the full-image Set-AT solutions run much faster than the multi-stage solutions, but they do not actually work, in terms of solving the problem, so the fact that they are fast is irrelevant. However, although accurate comparisons are difficult, these timings do allow us to make general observations about the relative performance of the two methods, and in particular allow us to show that the execution times of the multi-stage solutions do not suffer due to their larger size.

In terms of object detection performance measured using the mean FOM on the unseen test set, there is little difference between the full image and multi-stage versions of the experiments. The differences in mean FOM represent only very small numbers of false positives or false negatives relative to the hundreds of targets in the datasets. The only major difference in performance is on Set-AT. The full-image approach failed to produce any solutions that came even close to solving the problem, due to the lack of flexibility discussed at the end of the previous chapter. The multi-stage approach was able to co-evolve solutions that achieved good performance on this extremely hard dataset.

In spite of the similarities in performance between the full image approach and the multi-stage approach, there is still a clear reason to prefer the multi-stage approach. The training time necessary for the multi-stage approach is significantly lower. Where the full image approach often takes in the order of hundreds of days of CPU time to train, the multi-stage approach trains in only a few hours. This pairing of the co-evolutionary system

Dataset	Training set FOM	Testing set FOM	Solution size	Features used	Generations	Training Time (hours)
set-t	1.0 (0.0)	1.0 (0.0)	1.40 (0.82)	1.0 (0.0)	1.2 (0.41)	0.064 (0.021)
set-tn	1.0 (0.0)	1.0 (0.0)	18.3 (3.10)	3.7 (0.59)	9.4 (3.8)	0.55 (0.22)
set-tcn	1.0 (0.0)	1.00 (0.0045)	9.70 (3.11)	5.0 (0.79)	8.1 (2.9)	0.48 (0.18)
set-tcns	1.0 (0.0)	0.97 (0.012)	5.80 (1.64)	3.3 (1.0)	6.6 (1.7)	0.36 (0.089)
set-tcnr	1.0 (0.0)	0.98 (0.012)	24.5 (3.99)	6.3 (1.0)	126.4 (5.4)	8.7 (0.75)
set-tcnr	0.93 (0.017)	0.84 (0.024)	26.6 (5.93)	7.4 (0.50)	147.1 (13)	13 (1.3)
set-kc	0.90 (0.030)	0.85 (0.039)	51.4 (8.98)	6.4 (1.1)	131.1 (12)	12 (1.5)
set-p	0.91 (0.031)	0.85 (0.040)	64.6 (15.4)	6.8 (1.3)	145.7 (15)	15 (1.7)
set-at	0.76 (0.048)	0.71 (0.047)	252.2 (57.2)	18.4 (2.8)	200.0 (0.0)	70 (2.2)

(a) Summary of multi-stage method results

Dataset	Training set FOM	Testing set FOM	Solution size	Features used	Generations	Training Time (hours)
set-t	1.0 (0.0)	1.0 (0.0)	3.6 (1.1)	1.2 (0.42)	1.6 (0.61)	108 (44.05)
set-tn	1.0 (0.0)	1.0 (0.0)	3.65 (1.1)	1.7 (0.75)	2.8 (1.5)	196 (104.9)
set-tcn	1.0 (0.0)	1.00 (0.008)	4.9 (1.3)	2.3 (0.89)	11 (2.1)	789 (148.0)
set-tcns	1.0 (0.0)	0.99 (0.020)	23.4 (9.6)	2.4 (0.89)	12 (2.9)	990 (263.3)
set-tcnr	0.99 (0.046)	0.90 (0.056)	30.4 (6.8)	4.2 (1.0)	34 (4.2)	3046 (406.0)
set-tcnr	0.98 (0.031)	0.92 (0.054)	30.3 (11)	3.4 (0.61)	26 (11)	2308 (1020)
set-kc	1.0 (0.0)	1.00 (0.008)	13.1 (4.2)	2.3 (0.48)	2.2 (0.42)	162 (32.38)
set-p	0.99 (0.035)	0.94 (0.050)	35.5 (6.5)	2.7 (0.65)	27 (8.8)	2476 (853.9)
set-at	0.078 (0.020)	0.063 (0.021)	49.7 (13)	8.5 (2.9)	40 (0.0)	4308 (254.9)

(b) Summary of full-image methods results, repeated from Table 5.2 for convenience

Table 6.2: (a) Summary of results for the multi-stage method. All figures are averages over 20 runs. Standard deviations are shown in parentheses. (b) shows full-image results for comparison.

Dataset	Full Image			Multi-stage		
	Tree Size	Features	ex. Time (s)	Tree Size	Features	ex. Time (s)
set-t	3.60 (1.1)	1.20 (0.42)	0.20 (0.029)	1.40 (0.82)	1.0 (0.0)	0.12 (0.006)
set-tn	3.65 (1.1)	1.70 (0.75)	0.20 (0.038)	18.3 (3.10)	3.7 (0.59)	0.26 (0.024)
set-tcn	4.90 (1.3)	2.30 (0.89)	0.23 (0.046)	9.70 (3.11)	5.0 (0.79)	0.20 (0.024)
set-tcns	23.40 (9.6)	2.35 (0.89)	0.29 (0.054)	5.80 (1.64)	3.3 (1.0)	0.15 (0.012)
set-tcnr	30.40 (6.8)	4.20 (1.0)	0.40 (0.052)	24.5 (3.99)	6.3 (1.0)	0.28 (0.036)
set-tcnr	30.30 (11)	3.35 (0.61)	0.36 (0.045)	26.6 (5.93)	7.4 (0.50)	0.26 (0.046)
set-kc	13.05 (4.2)	2.30 (0.48)	0.25 (0.026)	51.4 (8.98)	6.4 (1.1)	0.39 (0.086)
set-p	35.45 (6.5)	2.70 (0.65)	0.34 (0.042)	64.6 (15.4)	6.8 (1.3)	0.45 (0.10)
set-at	49.70 (13)	8.50 (2.9)	0.67 (0.14)	252.2 (57.2)	18.4 (2.8)	1.2 (0.31)

Table 6.3: Summary of the size of the solutions generated by the full image and multi-stage techniques, and the run time of those solutions on a single image(in seconds). Standard deviations are shown in parentheses.

and the multi-stage approach allows extremely powerful object detection algorithms to be co-evolved quickly and without any domain knowledge.

6.5.1.1 FROC Analysis

In any object detection problem there is always a trade-off between detecting the targets and producing false positives. A detector could easily hit all targets by putting a guess at every pixel, but would incur an inordinate number of false positives. Likewise a detector could produce no false positives by making no guesses, but would therefore also miss all of the targets. The desired balance between these two factors is dependent on the problem at hand. In medical applications, for example, it is more important not to miss too many targets, even if that means producing false positives. In other applications false positives could be costly and minimising them is more important than hitting every single target. For example if this was a targeting system military application, then a single false positive could be disastrous.

This trade-off can be better understood through the use of a Receiver Operating Characteristic (ROC) analysis [28], which produces a curve of detection rate versus false-positive rate. ROC curves characterise the performance of the classifier so that the best settings

can be found for particular applications and allow classifiers to be compared. Normally, the ROC curve plots sensitivity (the proportion of all positive examples correctly classified) against specificity (the proportion of all negative cases correctly classified), which clearly requires that the bounds of each are known. In applications (such as object detection) where the false positive count is technically unbounded, a *free-response ROC* (FROC) curve is drawn which shares the same basic aims as the standard ROC curve but the FP values are no longer normalised.

As the Set-AT problem is the most difficult of the datasets, it makes an interesting case on which to perform this sort of detailed analysis. In order to calculate a FROC curve, we need to produce output from the system at various points along the sensitivity/specificity trade off line. In some systems this can be achieved by altering the threshold at which the classification decision is made. In this case we need to run the system multiple times with different parameters. Specifically, we change the β parameter of fitness function for the specialist stage evolution.

In order to generate the curve, a well performing primary stage solution is evolved (the primary stage solution shown in Figure 6.13 is used in this case). This solution is then used as the starting point for many different runs of the system with β values ranging from 1.0 to 20.0 (with α fixed at 1.0). Each of these runs produces a detection rate and false positive count (the single FOM figure is not appropriate here as the two separate values are needed) which gives a point in the FROC space. The input images are different sizes, and in order to produce a more intuitive graph, the false positive count is normalised by image area. The size of the image area used is also chosen to be intuitive and instead of being measured in pixels, it is measured in terms of the area that one hundred cars would occupy (hundred car area or HCA). Each car in the image occupies approximately 300 pixels, so one hundred cars would cover approximately 30,000 pixels. Each false positive count is therefore normalised using this figure.

The complete FROC space can be seen in Figure 6.19. A line of best fit through the points is used to create an approximation to the FROC curve. The uncertainties shown by the error bars are calculated as described in [41]. The FROC curve enables us to make decision about how the system should be used. If we require that all targets are detected, then the curve shows us that in order to achieve this, we would have to tolerate

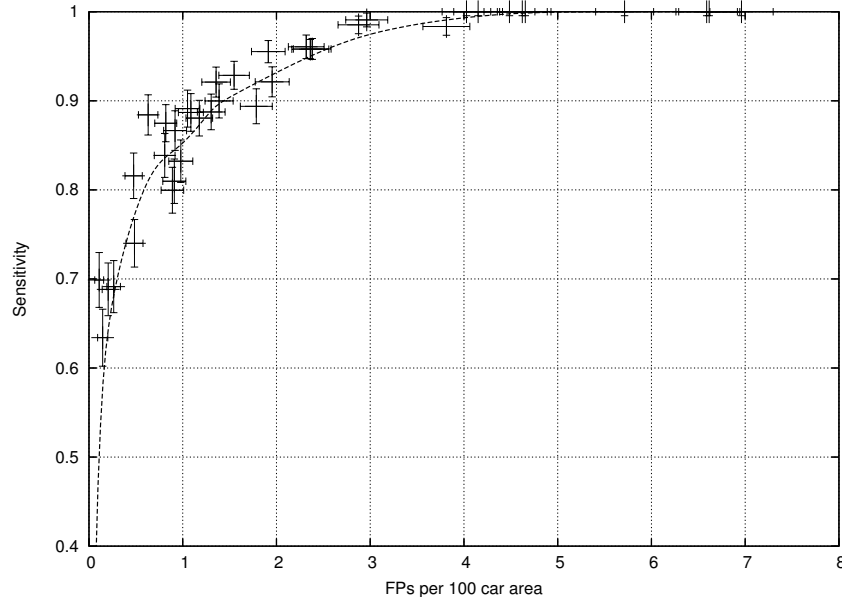


Figure 6.19: FROC curve for the set-AT problem. Note the y-axis commences at 0.4, not 0.0.

approximate 4 false positives per HCA. This would be achieved by setting β to around 4.0. If minimising false positives is vital, then the curve shows we would have to accept hitting 80% or less of the targets, and set β to a value of 19 or higher. Often, neither is more important and an equal balance is preferred. This would be represented by the closest point to the top left corner of the graph. In this case the point at (0.63,0.88) is closest, which was generated by setting the β parameter to 16.5

6.5.2 Effect of Feature Set Size

The two main reasons for attempting to co-evolve feature sets are as follows.

1. Arbitrarily chosen features may not extract information which is useful in solving the problems.
2. Having too many or too few features will make the problem harder to solve.

The first of these points is solved by using the co-evolutionary framework to allow the features to adapt, enabling them to return more useful information. The second point is more difficult to solve as it is impossible to tell for a particular problem how many features

are necessary to solve it [54]. In a fixed feature system, it would be difficult to determine how many features are necessary as the features are not generic - they are designed by humans. In a co-evolving system however, the features can adapt and you could assess and compare the effectiveness of the system using different numbers of features.

In the ideal case, the number of features would be able to change during the run, but this would introduce a number of issues outside of the scope of this thesis (some discussion of this issue can be found in Section 7.4.1). Currently however, only the maximum number of features can be specified, but this still enables us to analyse the performance of the co-evolutionary system with regard to feature set size.

An experiment was carried out using Set-P and Set-AT. The maximum number of features per stage was set to values from 1 to 10. For each of these values, 20 training runs were performed and the mean FOM plotted. This produces the graph shown in Figure 6.20. The results are as would intuitively be expected. Performance is poor when very few features are used, and then peaks, before gradually decreasing. This decrease in performance as the number of features increases is illustrative of the very purpose of feature optimisation techniques - learning is much harder when there is too much information. As the number of features increases the learner suffers from “information overload” finding it harder and harder to determine what information is actually useful to it for solving the problem.

6.5.3 Comparison to Fixed Statistic Method

The multi-stage method has been shown by others to be an effective method for tackling complex object detection problems. The important message of this chapter is to illustrate that the multi-stage method can be used to complement and enhance the abilities of the co-evolutionary architecture. In order to test the influence of the co-evolution, a series of experiments were performed in which the feature set was fixed and not evolving i.e. a fixed statistic system as described in Section 2.5.4. The Set-AT problem was chosen for this comparison as it is the most challenging and realistic of the attempted problems.

The set of 20 features shown in Figure 2.7 was used for this experiment. This feature set was fixed and did not have the ability to adapt through evolution as it does in the co-evolutionary framework. The multi-stage approach was applied in the same way as for

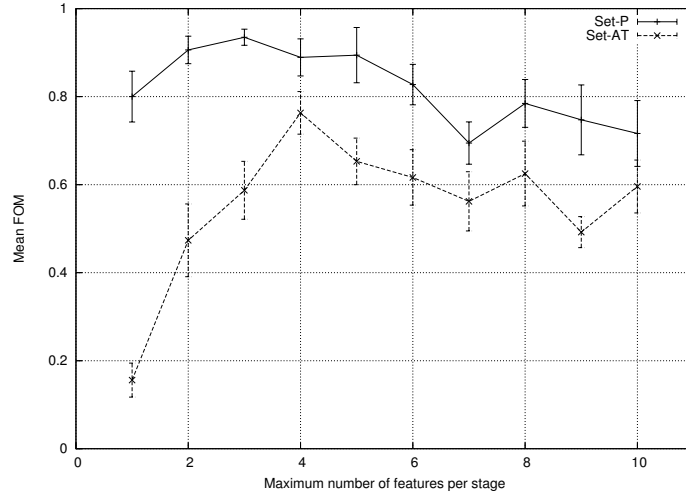


Figure 6.20: Average FOM produced on Set-P and Set-AT using different sizes of feature set.

the other experiments described earlier in this chapter. The best solution evolved had a training FOM of 0.37 and a test FOM of 0.31. This is very poor performance and did not constitute a useful solution to the problem.

In order to further assess the fixed statistic method, a FROC analysis was performed using the outputs of multiple runs with different β parameters. The resulting FROC curve is shown in Figure 6.21 alongside the curve for the co-evolutionary version. It can be seen that the co-evolutionary system is superior over the entire range. The fixed feature set results in much lower detection rates and higher false positives. Interestingly, both methods used approximately the same number of features in total. The multi-stage approach used a maximum of five features in each of four stages, giving up to 20 features in total. The feature set used for the fixed-set experiments also used 20 features. The only difference between the two was the co-evolution that allowed the features to adapt. The increase in performance can therefore confidently be attributed to the use of co-evolutionary feature extraction.

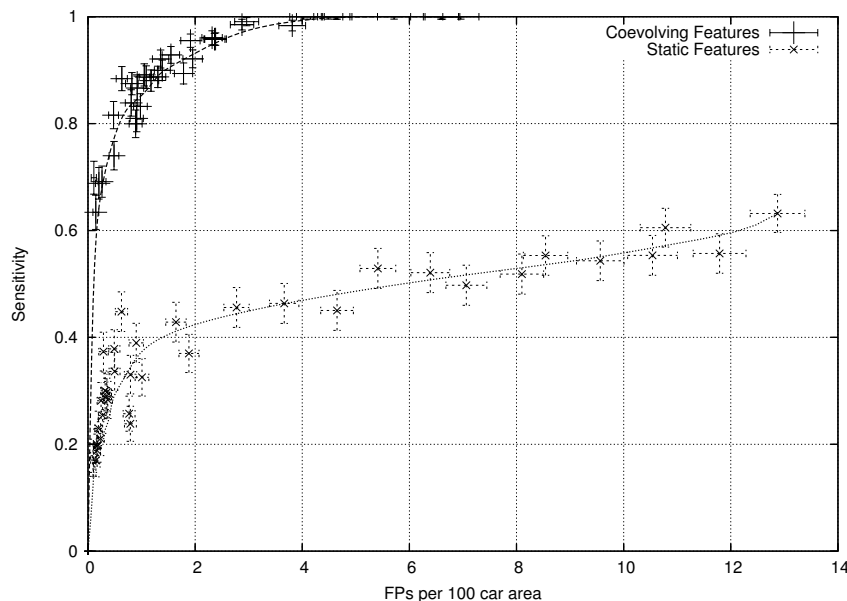


Figure 6.21: FROC curves showing fixed and co-evolving features. The co-evolving features show significantly better performance over the entire range.

6.6 Summary

This chapter has shown how a multistage approach to object detection can be used within the co-operative co-evolutionary framework. This allows the benefits of the multi-stage approach to be supplemented by the benefits of the co-evolutionary architecture. These two systems together can quickly create high quality object detection solutions on a wide range of synthetic and real-world imagery, using no knowledge of the problem domain.

The multi-stage approach was explained and then enhanced using a novel non-uniform sample selection scheme. Experiments were carried out on the datasets and performance on most of these datasets was shown to be equal to or better than those achieved using the full-image method. This is more impressive considering that the multi-stage technique uses only a fraction of the computation and works better on more complex data. Further analysis showed the effect of the feature set size on training.

A comparison to fixed feature set methods was performed and clearly showed that the use of co-evolutionary feature extraction produces significantly increased performance.

Chapter 7

Conclusions

The starting point of this thesis was the question of whether or not it is possible to co-evolve the feature extractors of an object detection system simultaneously with the detection system itself. This was an important question to study as evidence of feasibility in this area could be easily mapped onto many different problem domains and learning techniques. The fact that the system presented here can, for these object detection problems, learn an *entire* pre-processing and classification system without any domain knowledge is testament to the power of the technique. Only very recently have other researchers started to investigate the possibility of learning the pre-processing and classification stages simultaneously, such as [63] in 2005.

To many people, the very fact that the system can learn despite the fact that its inputs are changing is impressive in itself. However, it should come as no surprise, as we see this interdependence between species all around us in nature, often resulting in elegant and powerful solutions to the most complex of problems. This work has tried to capture the essence of that power, and although limited to a specific type of problem, the techniques developed and lessons learned should prove useful in applying the techniques to different problems, representations and learning techniques.

7.1 Summary

The work shown in this thesis has illustrated the use of co-evolutionary feature extraction in the domain of visual object detection tasks. Although this is a simple task within the

wider scope of computer vision systems, object detection represents an important sub-task, and currently one of particular interest in many fields.

It has been shown that the majority of current evolved object detection systems rely on hand crafted sets of features as inputs to their learning stage. These feature sets are often not adequate for solving the tasks, and are often too large, sometime even to the point of defeating the purpose of using a pre-processed feature set in the first place. This work has shown that instead of using these fixed feature sets, we can use co-evolution to evolve the feature sets *alongside the evolving detectors that are using them*.

Chapter 4 showed how these evolvable features could be represented, modified and efficiently calculated, and describe the co-evolutionary framework in which they could evolve alongside the detectors. Chapter 5 showed experiments which used a traditional style of evaluation with the co-evolutionary framework, which produced good results across a variety of problems which were equal to or significantly better than a fixed feature set approach in terms of accuracy. However, this method suffered from a lack of flexibility and scalability. Chapter 6 showed how a multi-stage evaluation could be used with the co-evolutionary framework in order to avoid these problems whilst still achieving excellent results. This method also required significantly less training time.

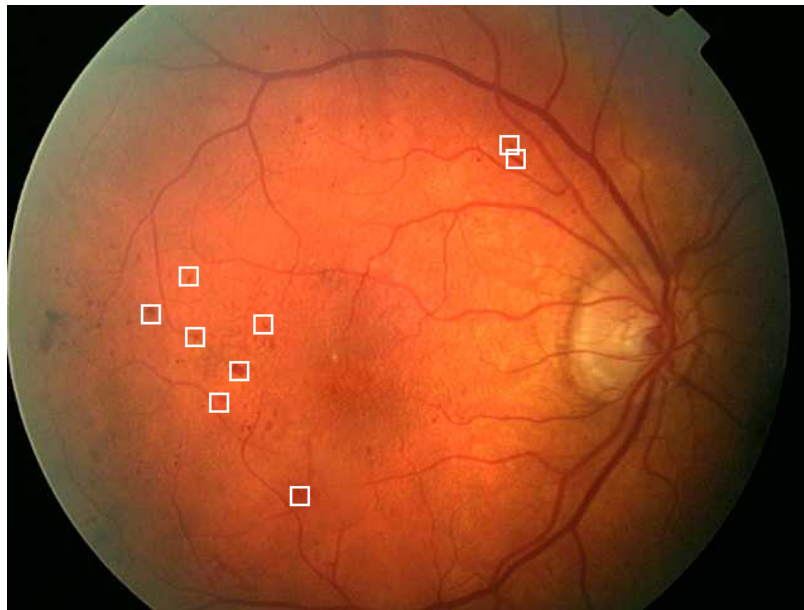
One of the major underlying aims in developing the co-evolutionary system is that the need for domain knowledge is removed. The human designed feature sets used in previous work often involve the introduction of specific or intuitive domain knowledge into the problem solver. This requires expertise in image analysis which precludes the use of these systems by domain experts who have no image analysis expertise. By learning the feature extraction stage as well as the classification stage, the need for domain knowledge is avoided, and the system can take in the actual raw data without any preprocessing. It could be argued that by specifying the types of features that the co-evolutionary system may use we are also providing domain knowledge. However, this knowledge is in the general *application domain*, of imaging, rather than the *problem domain* such as triangle detection. Application domain knowledge will always be present to some degree as it is inherent in the form of the raw data, but specifying what types of features are available is very different from specifying the exact features to use.

As both stages can be learned the co-evolutionary system could be used by domain experts with no knowledge of image processing or evolutionary computation. All that is required is for the expert to provide a dataset of images and binary masks showing where the targets are. For example, a doctor could supply a set of retinal images on which he has marked haemorrhages, which can be a significant problem in diabetes. The system could then learn to spot these without ever being told what they look like or what information might be useful for identifying them. This domain independence is reason enough to warrant detailed further analysis of these techniques. Preliminary work in this domain has met with some success but more comprehensive training data is needed to tackle the high FP rates. Currently a small dataset of 12 images is being used, on which the haemorrhages have been marked by an ophthalmologist. Experiments are performed using the multi-stage method as described in Chapter 6. Example output from one of the best detectors on an unseen test image is shown in Figure 7.1.

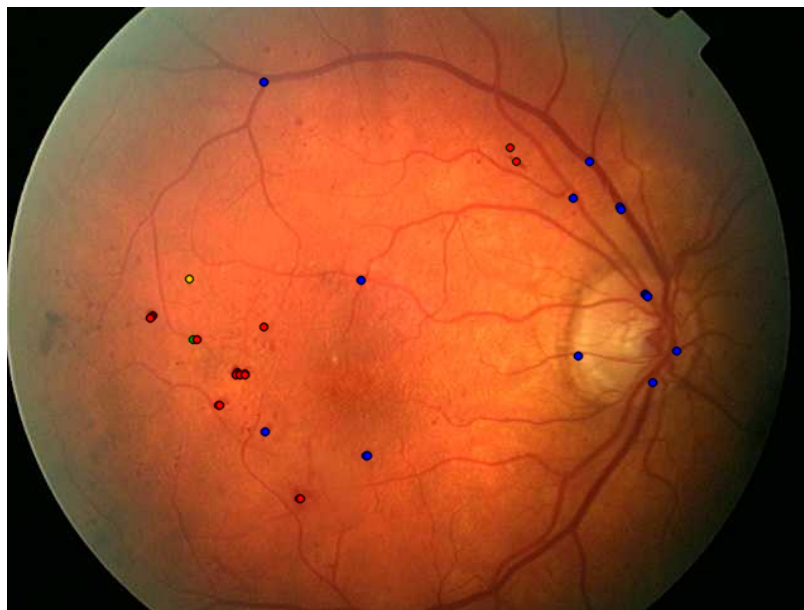
7.2 Contributions

This thesis has made a number of contributions to the field. Firstly a detailed literature survey was produced illustrating the use of evolutionary techniques in object detection problems. This review highlighted the important distinction between learning detectors and learning feature extractors. Using this distinction made it clear that previous work in the field uses learning in only one of the two stages. The problems associated with fixed feature sets were also highlighted as important issues. These facts lead to an important contribution noting that previous work in the field has not considered the possibility of learning both stages simultaneously.

One of the biggest contributions of this work is the design and development of a co-evolutionary architecture to allow the simultaneous evolution of features extractors and detectors. Firstly, this required the development of a representation for feature extractors along with appropriate evolutionary operators. We described an architecture for the co-evolutionary system, and described the underlying mechanisms of collaboration that allow solutions to be constructed, evaluated and evolved. A detailed discussion of the implementational issues involved in co-evolving features and detectors was also provided.



(a) Original image



(b) System output

Figure 7.1: An example of preliminary work on retinal haemorrhage detection. (a) Shows an unseen test image with ground-truth lesions marked. (b) Shows the output of a multi-stage co-evolved detector. The colour scheme is the same as previously used. There is a relatively high number of false positives (blue) compared to the number of targets.

A separate minor contribution of this thesis was the construction of a set of synthetic and natural image datasets for testing object detection problems. These test specific properties of object detections system and are available online for other researchers to use¹.

We demonstrated how this co-evolutionary architecture could be used with a traditional full image approach to object detection. A novel distance based fitness function was developed for this purpose which overcomes the shortcomings of typical functions used in the literature. This function provides a much smoother fitness landscape than traditional discreet techniques which produce very little selective pressure towards good solutions. This fitness function is separate from the co-evolutionary system and could also be applied in other evolutionary object detection systems. The function also included the notion of an *intuitive* penalty function for object detection problems, allowing negative situations to be penalised to the same degree as other comparable situations.

A detailed set of experimental results was produced, illustrating the success of the co-evolutionary approach and full image method on each of the datasets. These showed that efficient, compact solutions can be evolved using this method. We showed that these solutions were significantly more accurate than a fixed feature set system. The limitations in terms of computation, flexibility and scalability of the full image approach to object detection, were analysed and useful insight was gained into the nature of this system.

We showed how a multi-stage approach to object detection [46] could be used within the co-evolutionary framework. This work was also one of the most thorough experimental investigation into the use of this multi-stage method, and has provided further evidence of its effectiveness. An improvement was made to the multi-stage method – in the first stage of the algorithm a sample of non-target points is selected. The selection method was observed to be sub-optimal, and a new approach was designed which produced a smaller more efficient sample.

A detailed set of experimental results was produced documenting the capabilities of the combined co-evolutionary/multi-stage approach. It was found that the multi-stage method produced similar performance to the full-image approach, but using only a fraction of the computation time. We also concluded that the multi-stage approach was more flexible

¹<http://www.cs.bham.ac.uk/~mer/datasets/>

and that this flexibility enabled it to adapt to more complex problems with relative ease. We showed how this increased performance was however accompanied by an increase in overall solution size.

In comparison to traditional approaches in which the feature set is hand crafted and fixed, the co-evolutionary approach was shown to have significantly better performance over a complete FROC curve.

A major conceptual contribution of this work is the discovery of the fact that it is possible to simultaneously learn the two stages of the classification pipeline. As discussed previously, the feature extraction phase rarely receives much attention even though it contains fundamental assumptions and biases that influence the subsequent stages. The traditional classification pipeline has been unchanged for many years, and the work in this thesis contributes an alternative view that may start to change the way in which we think about these types of learning problems.

Although in this work evolutionary computation techniques were used for the feature extraction and classification stages, there is no reason why other techniques could not be used. The findings of this thesis lay the foundations for further investigation into generalised co-evolutionary pre-processor and classifier learning, which has the potential to be a significant advance in the field of machine learning. For further discussion about this issue, see section 7.4.3.

7.2.1 Contributions of the System

It is only fair to give some credit to the contributions made by the co-evolutionary system. The solutions that were co-evolved during the experimental phase of this thesis are very clever designs in their own right. Many of the solutions produced would probably be comparable in terms of accuracy and efficiency with those produced by a human designer. Further analysis of these and other solutions could provide useful insight into the development of efficient real-time object detectors, and for this reason, the contributions of the system deserve recognition here. In particular, the reinvention of previously human-invented algorithms, such as the adaptive threshold algorithm evolved for Set-KC in Section 5.6.7 qualifies the solution for entry into the annual Human-Competitive Result competition, which rewards evolutionary algorithms that produce *“results that are not*

merely interesting and impressive, but competitive with the work of creative and inventive humans”.

7.3 Limitations

There are several limitations to the system which should be mentioned here. One of the most notable is the simplicity of the feature representation. Efficiency of evaluation played a large part in the choice of this representation and it would be naïve to assume that these simple statistics can capture highly useful information from a diverse range of images. This issue is discussed further in the next section. The second feature related limitation is the fact that maximum number of features has to be manually set. If the size of the feature set was dynamic then the system would be free of one of its most significant human-set parameters. This issue is discussed further in Section 7.4.1.

A limitation of the work in general is that only a small number of real world datasets were used in the experiments. Although the results were promising, it is difficult to confidently predict the system’s behaviour on a wide range of different image datasets. This is something of a problem throughout the field, where the availability of high quality datasets and accompanying ground-truths is poor.

Further analysis is also required into the performance gains that the co-evolutionary approach gives over fixed feature systems. The analysis in Section 6.5.3 clearly shows that the co-evolutionary system is superior on Set-AT. Prohibitive computation costs mean that this sort of detailed analysis is not available for other datasets.

Although demonstrating feasibility and superiority over fixed feature systems were the major aims of this thesis, the lack of comparison to non-evolutionary techniques is acknowledged as a limitation. There is a large body of work using non-evolutionary techniques for the automatic creation and learning of object detectors and it would be prudent and interesting to compare the co-evolutionary system against several of the current favourites of the computer vision community, such as a Viola and Jones style classifier cascade [118] and Lowe’s SIFT system [69]. Comparing the system against these two techniques would give more insight into its capabilities in terms of training time and performance. The Viola-Jones system, for example is much faster to train, and execution times would be similar, but it cannot deal with rotational variance of the targets, like the

co-evolutionary system can. The SIFT system on the other hand is invariant to most affine transformations including scale and rotation, and is also able to detect occluded objects. This would make an interesting comparison as theoretically, the multi-stage technique could detect occluded objects due to the way that the different stages focus on different parts of the target. Performing these experimental comparisons would help to answer the question of whether the extra training time required for the co-evolutionary approach is worthwhile in terms of accuracy and execution time of the solutions.

There are of course, many other object detection methodologies, and experimental comparisons against these would be useful. However, the field suffers from a lack of standard test problems, and therefore current benchmarks, to compare against. This is a problem in many subfields of AI, but seems to be more of a problem in computer vision where the ground truth data available often does not match with the requirements of a particular algorithm. This leads to the creation of custom datasets which are useful for the comparison of only a very small class of algorithms.

7.4 Future Work

7.4.1 Dynamic Number of Features

One of the major aims of this work was to create a system that did not require domain knowledge in order to evolve a solution. This has been achieved through the use of the co-evolutionary framework, but in doing so a different set of parameters has been introduced. One of the most significant parameters is the maximum feature set size, which determines how many features (and therefore feature populations) are used in the system. As with any learning problem, we want to use as few features as possible to solve the problem, so it is unfortunate that this parameter must be manually set to a fixed value. This is a maximum value and there is no compulsion on the solution to use all of the available features, and indeed the solutions very often do use less. However, the fact that this parameter needs to be set manually detracts from the high levels of adaptability elsewhere in the system.

Ideally, the number of features *available* (not necessarily used) would change throughout the training run. It would increase if more features were necessary to solve the problem,

and would decrease if there were surplus features which were not necessary or confusing. Achieving these aims is non-trivial. Firstly there are a number of issues regarding how we could measure whether there are too few or too many features. For example, as the features are changing, it is difficult to tell whether there are too few features, or if the features that we do have are just not returning useful information.

Assuming some mechanism for measuring if new features are needed, or features need to be removed, there are more complex issues with regard to actually making the change. Adding a feature would be a simple operation. A new random feature population could be added and a new terminal added to allow its use. Deleting features would be much more troublesome as the detectors may be referencing them. If a feature was deleted then the trees that refer to it need to be modified in some way. There are several possible courses of action, but further investigation would be required to determine an acceptable method.

7.4.2 Generic Features

The pixel-statistic features used in the system have shown that they are extremely powerful, but in essence they are very simplistic and are useful more for their speed than for their utility in solving problems of this type. More complex problems would greatly benefit from the ability to use more expressive features. However, more expressive features come at a huge computational cost. For example, it would be possible to use a large evolving convolution kernel as a feature extractor. This would have enormous power and could capture information from the image many times more useful than the basic rectangular features. Convolution, however, is a very expensive operation, and although performance gains can be achieved using Fourier space convolution, for example, the computation time would still be prohibitive. However, as more accelerated graphics hardware becomes available, and APIs allow easy access to it, this prohibitive computation could be carried out in hardware, reducing the overhead and enabling these more expressive features to be used freely.

Even though convolution is a powerful operation it is still not the ideal choice for a feature representation. Convolution kernels tend to be very inflexible as each of their values is fixed, very specific, and localised at an exact point. There is no “fuzziness” that would allow the features to be more tolerant of noise and have the flexibility to deal with

variation in both targets and background. Also, convolution is a linear operation which limits its capabilities to only a relatively small subset of possible outcomes. Experiments into evolving non-linear operations such as those used in mathematical morphology are beginning to show promise [38, 5], and may be a useful future direction.

A representation is needed that has both the power to represent complex structures but also a degree of fuzziness, modularity and adaptability. For practical reasons, and especially so for evolutionary purposes, the representation also needs to be computable in a reasonably small amount of time. Such a representation would be extremely valuable in many fields and application areas and is a goal worth pursuing.

7.4.3 A Generic Co-evolutionary Preprocessor

The methods developed in this work have a huge potential to be applied to different representations, learning methods and applications. The vast majority of learning problems have a two-stage architecture as described in Section 2.3, where a pre-processing stage extracts key features from the raw data and then passes them to a classification stage in order to produce the final class labels. Any method that could tackle both of these stages simultaneously as one problem would be useful in many domains.

All that is required in order to apply this technique to a two-stage problem is a suitable representation and operators for the feature extractors, and for the classifiers. Pixel-statistic based features were used here only because the raw data was in the form of images. In other domains, different feature extraction schemes would be used.

7.4.3.1 Example Application

In preliminary work to test the applicability of the system, we have applied the co-evolutionary framework to a problem other than object detection. In fact, the problem is not even a classification problem, but a regression style problem in which a model and its inputs must be created to fit experimental data. The problem is that of parameter recovery from fluorescence microscopy images [18], which is essentially a two-stage process. In the first stage, images of the sample are acquired through a set of spectral filters². The

²Although images are used, this problem is not strictly an imaging problem as spatial properties and dependencies are not present. Each pixel is treated as an independent training case.

outputs from the filters are passed to a second stage which attempts to recover the composition of the original sample. These two stages map neatly onto the stages used in the co-evolutionary architecture. The filtering is equivalent to the feature extraction stage, which tries to extract useful information from the raw data. As the filters are Gaussian curves, they can be easily represented as two real values. These values are evolved using a standard real-valued optimisation technique, Fast Evolutionary Programming (FEP) [123]. In the second stage an expression must be created in terms of the filtered values which outputs several parameters representing the composition of the sample. As this is a multi-output problem, tree based GP could not be used, so a different form of GP, called Cartesian GP [72] was used. A simple multi-objective fitness function was used to minimise the error between the system's outputs and the ground-truth. In spite of these changes to the problem domain and implementation of the stages, no changes were necessary to the co-evolutionary framework and it was used as described in Chapter 4. This is an encouraging result and shows that there may be many other areas in which a generic co-evolutionary pre-processor may be applicable.

7.5 Concluding Remarks

Like many other sub-fields in machine learning, the area of evolving visual object detection systems has not changed much since its initial inception. Minor advances have been made in the particular learning methods and processes used, but there have been no fundamental changes to the way that the problem is approached for a long time. The work presented in this thesis takes a fresh look at the problem as a whole, and in doing so shows how we can use newer ideas (such as co-operative co-evolution) to fundamentally change the way that we think about the problem. The results shown here are by no means comprehensive, but they do show that this area is worthy of detailed further study.

In an attempt to solve ever more complex problems, we often look towards ever more complex solutions. If we instead look towards nature we see that the opposite is true and complex problems are solved not by complex solutions but rather by the complex interaction between simple solutions. Co-operative co-evolution is a prime example of this and this work has attempted to show that this power can be exploited in order to redefine

how we think evolving object detection solution, and indeed two-stage classification in general.

References

- [1] Manu Ahluwalia and Larry Bull. Coevolving functions in genetic programming. *Journal of Systems Architecture*, 47(7):573–585, July 2001.
- [2] Manu Ahluwalia, Larry Bull, and Terence C. Fogarty. Co-evolving functions in genetic programming: A comparison in ADF selection strategies. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 3–8, San Francisco, CA, USA, 13-16 1997. Morgan Kaufmann.
- [3] David Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 23, pages 477–494. MIT Press, 1994.
- [4] Shinya Aoki and Tomoharu Nagao. Automatic construction of tree-structural image transformation using genetic programming. In *Proceedings of the 1999 International Conference on Image Processing (ICIP-99)*, volume 1, pages 529–533, Kobe, October 24–28 1999. IEEE.
- [5] Lucia Ballerini and Lennart Franzén. Genetic optimization of morphological filters with applications in breast cancer detection. In Guenther R. Raidl, Stefano Cagnoni, Jurgen Branke, David W. Corne, Rolf Drechsler, Yaochu Jin, Colin Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*,

- volume 3005 of *LNCS*, pages 250–259, Coimbra, Portugal, 5-7 April 2004. Springer Verlag.
- [6] Richard E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, NJ, 1961.
- [7] Tony Belpaeme. Evolution of visual feature detectors. In Riccardo Poli, Stefano Cagnoni, Hans-Michael Voigt, Terry Fogarty, and Peter Nordin, editors, *Late Breaking Papers at EvoIASP'99*. University of Birmingham Technical Report CSRP-99-10, 1999.
- [8] Bir Bhanu and Krzysztof Krawiec. Coevolutionary construction of features for transformation of representation in machine learning. In Alwyn M. Barry, editor, *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 249–254, New York, 8 July 2002. AAAI.
- [9] Bir Bhanu and Yingqiang Lin. Object detection in multi-modal images using genetic programming. *Applied Soft Computing*, 4(2):175–201, May 2004.
- [10] W. Ekkehard Blanz, Charles E. Cox, and Sheri L. Gish. *Connectionist Architectures in Low Level Image Segmentation*, pages 883–920. Wolrd Scientific Press, 1993.
- [11] Bradley J. Bozarth. Programmatic compression of video using genetic programming. In John R. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford 2000*, pages 46–53. Stanford Bookstore, Stanford, California, 94305-3079 USA, June 2000.
- [12] R.N. Bracewell. *The Fourier Transform and its Applications*. McGraw Hill, New York, 1965.
- [13] Steven P. Brumby, Neal R. Harvey, Jeffrey J. Bloch, James P. Theiler, Simon J. Perkins, Aaron C. Young, and John J. Szymanski. Evolving forest fire burn severity classification algorithms for multispectral imagery. In Sylvia S. Shen and Michael R. Descour, editors, *Proceedings of SPIE, Algorithms for Multispectral, Hyperspectral, and Ultraspectral Imagery VII*, volume 4381, pages 236–245. SPIE, August 2001.

- [14] Steven P. Brumby, Neal R. Harvey, Simon J. Perkins, Reid B. Porter, John J. Szymanski, James Theiler, and Jeffrey J. Bloch. A genetic algorithm for combining new and existing image processing tools for multispectral imagery. In Sylvia S. Shen and Michael R. Descour, editors, *Proceedings of SPIE Algorithms for Multispectral, Hyperspectral, and Ultraspectral Imagery VI*, pages 480–490, August 2000.
- [15] Steven P. Brumby, James Theiler, Jeffrey J. Bloch, Neal R. Harvey, Simon Perkins, John J. Szymanski, and A. Cody Young. Evolving land cover classification algorithms for multispectral and multitemporal imagery. In Michael R. Descour and Sylvia S. Shen, editors, *Proc. SPIE Imaging Spectrometry VII*, volume 4480, 2002.
- [16] Steven P. Brumby, James Theiler, Simon Perkins, Neal R. Harvey, and John J. Szymanski. Genetic programming approach to extracting features from remotely sensed imagery. In *FUSION 2001: Fourth International Conference on Image Fusion*, Montreal, Quebec, Canada, 7-10 August 2001.
- [17] C.K. Chow and T. Kaneko. Automatic boundary detection of the left ventricle from cineangiograms. *Computers and Biomedical Research*, 5:388–410, 1972.
- [18] Ela Claridge, Dale J. Powner, and Michael J.O. Wakelam. The analysis of fluorescence microscopy images for fret detection. In Majid Mirmehdi, editor, *Proceedings of Medical Image Understanding and Analysis (MIUA) 2005*, pages 263–266, Bristol, UK, 2005.
- [19] Michael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In John J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, 1985.
- [20] Franklin C. Crow. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 207–212, New York, NY, USA, 1984. ACM Press.
- [21] A. Winklhofer D. Dickmanns, J. Schmidhuber. Der genetische algorithmus: Eine implementierung in prolog. Technical report, Technical University of Munich, Institut f. Informatik, 1987.

- [22] Jason M. Daida, Tommaso F. Bersano-Begey, Steven J. Ross, and John F. Vesecky. Computer-assisted design of image classification algorithms: Dynamic and static fitness evaluations in a scaffolded genetic programming environment. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 279–284, Stanford University, CA, USA, 28–31 1996. MIT Press.
- [23] Anthony B. Davis, Steven P. Brumby, Neal R. Harvey, Karen Lewis Hirsch, and Charles A. Rohde. Genetic refinement of cloud-masking algorithms for the multi-spectral thermal imager (MTI). In *Proceedings of IEEE International Geoscience and Remote Sensing Symposium 2001*, Sydney, Australia, 9-13 July 2001. IEEE.
- [24] Guido C. H. E. de Croon, Eric O. Postma, and H. Jaap van den Herik. Sensory-motor coordination in gaze control. In Franz Rothlauf, Juergen Branke, Stefano Cagnoni, David W. Corne, Rolf Drechsler, Yaochu Jin, Penousal Machado, Elena Marchiori, Juan Romero, George D. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3449 of *LNCS*, pages 334–344, Lausanne, Switzerland, 30 March-1 April 2005. Springer Verlag.
- [25] Damian R. Eads, Steven J. Williams, James Theiler, Reid Porter, Neal R. Harvey, Simon J. Perkins, Steven P. Brumby, and Nancy A. David. A multimodal approach to feature extraction for image and signal learning problems. In Bruno Bosacchi, David B. Fogel, and James C. Bezdek, editors, *Proceedings of the 6th SPIE Conference on the Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation*, volume 5200, pages 79–90, December 2003.
- [26] Marc Ebner. On the evolution of interest operators using genetic programming. In Riccardo Poli, W. B. Langdon, Marc Schoenauer, Terry Fogarty, and Wolfgang Banzhaf, editors, *Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming*, pages 6–10, Paris, France, 14-15 April 1998. CSRP-98-10, The University of Birmingham, UK.

- [27] Marc Ebner and Andreas Zell. Evolving a task specific image operator. In Riccardo Poli, Hans-Michael Voigt, Stefano Cagnoni, Dave Corne, George D. Smith, and Terence C. Fogarty, editors, *Evolutionary Image Analysis, Signal Processing and Telecommunications: First European Workshop, EvoIASP'99 and EuroEcTel'99*, volume 1596 of *LNCS*, pages 74–89, Goteborg, Sweden, 28-29 May 1999. Springer-Verlag.
- [28] James P. Egan. *Signal Detection Theory and ROC Analysis*. Academic Press, New York, 1975.
- [29] Herman Ehrenburg. Improved direct acyclic graph handling and the combine operator in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 285–291, Stanford University, CA, USA, July 1996. MIT Press.
- [30] Alex Fukunaga and Andre Stechert. Evolving nonlinear predictive models for lossless image compression with genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 95–102, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.
- [31] Alvara Guardo, Christophe Le Gal, and Augustin Lux. Evolving visual features and detectors. In Luciano da Fontoura and Gilberto Camara, editors, *International Symposium on Computer Graphics, Image Processing, and Vision (SIGGRAPPI 98)*, pages 246–253, Rio De Janeiro, Brazil, October 1998. IEEE.
- [32] Peter J.B. Hancock. An empirical comparison of selection methods in evolutionary algorithms. In Terry C. Fogarty, editor, *Proceedings of AISB Evolutionary Computing Workshop*, volume 865 of *Lecture Notes in Computer Science*, pages 80–94, Leeds, U.K., 1994. Springer-Verlag.

- [33] S. Handley. On the use of a directed acyclic graph to represent a population of computer programs. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pages 154–159, Orlando, Florida, USA, June 1994. IEEE Press.
- [34] Neal R. Harvey, Steven P. Brumby, Simon J. Perkins, Reid B. Porter, James P. Theiler, Aaron C. Young, John J. Szymanski, and Jeffrey J. Bloch. Parallel evolution of image processing tools for multispectral imagery. In Michael R. Descour and Sylvia S. Shen, editors, *Proceedings of SPIE - Imaging Spectrometry VI*, volume 4132, pages 72–82. SPIE, November 2000.
- [35] Neal R. Harvey, Simon J. Perkins, Steven P. Brumby, James Theiler, Reid B. Porter, A. Cody Young, Anil K. Varghese John J. Szymanski, and Jeffrey J. Bloch. Finding golf courses: The ultra high tech approach. In Stefano Cagnoni, editor, *Proceedings of the Second European Workshop on Evolutionary Computation in Image Analysis and Signal Processing (EvoIASP)*, pages 54–64, 17April 2000.
- [36] Neal R. Harvey, James Theiler, Steven P. Brumby, Simon Perkins, John J. Szymanski, Jeffrey J. Bloch, Reid B. Porter, Mark Galassi, and A. Cody Young. Comparison of GENIE and conventional supervised classifiers for multispectral image feature extraction. *IEEE Transactions on Geoscience and Remote Sensing*, 40(2):393–404, February 2002.
- [37] John M. Henderson. Human gaze control during real-world scene perception. *Trends in Cognitive Sciences*, 7(11):498–504, November 2003.
- [38] Marcos I. Quintana Hernández. *Genetic Programming Applied to Morphological Image Processing*. PhD thesis, School of Computer Science, University of Birmingham, 2005.
- [39] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial life II*, volume 10 of *Sante Fe Institute Studies in the Sciences of Complexity*, pages 313–324. Addison-Wesley, Redwood City, Calif., 1992.

- [40] Hong S. Hong. Digital image restoration using genetic programming. In John R. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford 1999*, pages 68–75. Stanford Bookstore, Stanford, California, 94305-3079 USA, 15 March 1999.
- [41] James Horne. Semi-parametric froc analysis of automated target recognition systems in satellite imagery. In *Proceedings of the Annual Conference of the American Society for Photogrammetry and Remote Sensing 2004*, Colorado, 2004. ASPRS.
- [42] Daniel Howard and Simon C. Roberts. Evolving object detectors for infrared imagery: a comparison of texture analysis against simple statistics. In Kaisa Miettinen, Marko M. Mkel, Pekka Neittaanmki, and Jacques Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 79–86, Chichester, UK, 30 1999. John Wiley & Sons.
- [43] Daniel Howard and Simon C. Roberts. A staged genetic programming strategy for image analysis. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1047–1052, San Francisco, CA 94104, USA, 13-17 1999. Morgan Kaufmann.
- [44] Daniel Howard, Simon C. Roberts, and Richard Brankin. Target detection in SAR imagery by genetic programming. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, Stanford, CA, USA, 22-25 1998. Stanford University Bookstore.
- [45] Daniel Howard, Simon C. Roberts, and Richard Brankin. Evolution of ship detectors for satellite SAR imagery. In Riccardo Poli, Peter Nordin, William B. Langdon, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP’99*, volume EuroGP’99, part of poli:1999:GP of 3-540-65899-8, pages 135–148, Berlin, 26-27 1999. Springer-Verlag.
- [46] Daniel Howard, Simon C. Roberts, and Richard Brankin. Target detection in imagery by genetic programming. *Advances in Engineering Software*, 30(5):303–311, 1999.

- [47] David H. Hubel. *Eye, Brain and Vision*. Scientific American Library, New York, 1988.
- [48] Michael Johnson, Pattie Maes, and Trevor Darrel. Evolving visual routines. In *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Things*, pages 198–209. MIT Press, July 1994.
- [49] Wolfgang Kantschik and Wolfgang Banzhaf. Linear-graph GP—A new GP structure. In James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 83–92, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.
- [50] Maarten Keijzer. Alternatives in subtree caching for genetic programming. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 328–337, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.
- [51] Maarten Keijzer, Conor Ryan, Michael O'Neill, Mike Cattolico, and Vladan Babovic. Ripple crossover in genetic programming. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 74–86, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.
- [52] Mike J. Keith and Martin C. Martin. Genetic programming in C++: Implementation issues. In Kenneth E. Kinneer, Jr., editor, *Advances in Genetic Programming*, chapter 13, pages 285–310. MIT Press, 1994.
- [53] Andreas Klappenecker and Frank U. May. Evolving better wavelet compression schemes. In Andrew F. Laine, Michael A. Unser, and Mladen V. Wickerhauser, editors, *Wavelet Applications in Signal and Image Processing III*, volume 2569, San Diego, CA, USA, 9-14 July 1995. SPIE.

- [54] Ranganath Kothamasu, Samuel H. Huang, and William H. VerDuin. Analysis of performance evaluation metrics to combat the model selection problem. In Elena R. Messina and Alex M. Meystel, editors, *Proceedings of Performance Metrics for Intelligent Systems (PerMIS)*. National Institute for Standards and Technology, 2003.
- [55] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [56] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.
- [57] John R. Koza, David Andre, Forrest H Bennett III, and Martin Keane. *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman, April 1999.
- [58] John R. Koza, Forrest H Bennett III, David Andre, and Martin A. Keane. Evolutionary design of analog electrical circuits using genetic programming. In *Proceedings of Adaptive Computing in Design and Manufacture Conference*, Plymouth, England, 21-23 April 1998.
- [59] John R. Koza, Forrest H Bennett III, David Andre, and Martin A. Keane. Automatic design of analog electrical circuits using genetic programming. In Hugh Cartwright, editor, *Intelligent Data Analysis in Science*, chapter 8, pages 172–200. Oxford University Press, Oxford, 2000.
- [60] John R. Koza, Martin A. Keane, and Matthew J. Streeter. Routine high-return human-competitive evolvable hardware. In Ricardo S. Zebulum, David Gwaltney, Gregory Horbny, Didier Keymeulen, Jason Lohn, and Adrian Stoica, editors, *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 3–17, Seattle, 24-26 June 2004. IEEE Press.
- [61] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

- [62] Thomas Krantz, Oscar Lindberg, Gunnar Thorburn, and Peter Nordin. Programmatic compression of natural video. In Erick Cantú-Paz, editor, *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 301–307, New York, NY, July 2002. AAAI.
- [63] Thomas Landgrebe, Pavel Paclk, David M. J. Tax, and Robert P. W. Duin. Optimising two-stage recognition systems. In N.C. Oza, R Polikar, J. Kittler, and F. Roli, editors, *Proceedings of Multiple Classifier Systems 2005*, volume 3541 of *Lecture Notes in computer Science*, pages 206–215. Springer, 2005.
- [64] W. B. Langdon. Size fair tree crossovers. In Eric Postma and Marc Gyssen, editors, *Proceedings of the Eleventh Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'99)*, pages 255–256, Kasteel Vaeshartelt, Maastricht, Holland, 3-4 November 1999.
- [65] William B. Langdon and Wolfgang Banzhaf. Repeated patterns in tree genetic programming. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 190–202, Lausanne, Switzerland, 30 March - 1 April 2005. Springer.
- [66] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In Lisboa P.G.J., editor, *Neural Networks, current applications*. Chappman and Hall, 1992.
- [67] Jason Lohn, Gregory Hornby, and Derek Linden. Evolutionary antenna design for a NASA spacecraft. In Una-May O'Reilly, Tina Yu, Rick L. Riolo, and Bill Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 18. Kluwer, Ann Arbor, 13-15 May 2004.
- [68] Jason D. Lohn, Derek S. Linden, Gregory S. Hornby, William .F. Kraus, Adaan Rodriguez-Arroyo, and S. E. Seufert. Evolutionary design of an X-band antenna for NASA's space technology 5 mission. In *Proceedings of the NASA/DoD Conference on Evolvable Hardware*, pages 155–163. IEEE, 2003.

- [69] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [70] Penousal Machado, Andre Dias, Nuno Duarte, and Amilcar Cardoso. Giving colour to images. In Amilcar Cardoso and Geraint Wiggins, editors, *AI and Creativity in Arts and Science*, Imperial College, United Kingdom, 2-5 April 2002. A symposium as part of AISB’02.
- [71] Paulo Menezes, José Barreto, and Jorge Dias. Human-robot interaction based on haar-like features and eigenfaces. In *International Conference on Robotics and Automation*, pages 1888–1893, New Orleans, May 2004. IEEE.
- [72] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP’2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15-16 April 2000. Springer-Verlag.
- [73] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
- [74] Cristian Munteanu and Agostinho Rosa. Evolutionary image enhancement with user behaviour modeling. In *SAC ’01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 316–320, New York, NY, USA, 2001. ACM Press.
- [75] Patrick Murphy. Uci repository of machine learning databases. <http://www.ics.uci.edu/mlearn/MLRepository.html>, 1994.
- [76] Peter Nordin. A compiling genetic programming system that directly manipulates the machine code. In Kenneth E. Kinneer, Jr., editor, *Advances in Genetic Programming*. MIT Press, 1994.
- [77] Peter Nordin and Wolfgang Banzhaf. Programmatic compression of images and sound. In John R. Koza, J. R. Golberg, David E. Fogel, and R. L Riolo, editors, *Genetic Programming 96: Proceedings of the First Annual Conference*, pages 345–350, Stanford University, California, 1996. MIT Press.

- [78] Johan Parent and Ann Nowe. Evolving compression preprocessors with genetic programming. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 861–867, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
- [79] Simon Perkins. *Incremental Acquisition of Complex Visual Behaviour using Genetic Programming*. PhD thesis, University of Edinburgh, 1998.
- [80] Simon Perkins. Evolving robot vision: Increasing performance through shaping. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 381–388, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.
- [81] Simon J. Perkins, James P. Theiler, Steven P. Brumby, Neal R. Harvey, Reid B. Porter, John J. Szymanski, and Jeffrey J. Bloch. GENIE: a hybrid genetic algorithm for feature classification in multispectral images. In Bruno Bosacchi, David B. Fogel, and James C. Bezdek, editors, *Proceedings of SPIE, Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation III*, volume 4120, pages 52–62. SPIE, October 2000.
- [82] Catherine S. Plesko, Steven P. Brumby, and Conway B. Leovy. Automatic feature extraction for panchromatic mars global surveyor mars orbiter camera imagery. In Michael R. Descour and Sylvia S. Shen, editors, *Proceedings of SPIE Imaging Spectrometry VII*, volume 4480. SPIE, January 2002.
- [83] Riccardo Poli. Genetic programming for feature detection and image segmentation. In T. C. Fogarty, editor, *Evolutionary Computing*, number 1143 in Lecture Notes in Computer Science, pages 110–125. Springer-Verlag, University of Sussex, UK, 1-2 April 1996.
- [84] Riccardo Poli. Genetic programming for feature detection and image segmentation. In Terry Fogarty, editor, *Proceedings of the AISB’96 Workshop on Evolutionary*

- Computation*, volume 1143 of *Lecture Notes in Computer Science*, pages 110–125. Springer, April 1996.
- [85] Riccardo Poli. Genetic programming for image analysis. Technical Report CSRP-96-1, University of Birmingham, UK, January 1996.
- [86] Riccardo Poli. Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, B15 2TT, UK, September 1996.
- [87] Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming. In Thomas Back, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 346–353, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann.
- [88] Riccardo Poli. Parallel distributed genetic programming. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, Advanced Topics in Computer Science, chapter 27, pages 403–431. McGraw-Hill, Maidenhead, Berkshire, England, 1999.
- [89] Riccardo Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In E. Costa C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, editor, *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003)*, volume 2610 of *LNCS*, pages 204–217, Essex, UK, 2003. Springer Verlag.
- [90] Riccardo Poli and Stefano Cagnoni. Evolution of psuedo-colouring algorithms for image enhancement with interactive genetic programming. Technical Report CSRP-97-5, School of Computer Science, The University of Birmingham, B15 2TT, UK, January 1997. Presented at GP-97.
- [91] M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer, 1994. Lecture Notes in Computer Science.

- [92] Mitchell A. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, George Mason University, 1997.
- [93] Mitchell A. Potter and Kenneth A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.
- [94] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.
- [95] Michael L. Raymer, William F. Punch, Erik D. Goodman, and Leslie A. Kuhn. Genetic programming for improved data mining - application to the biochemistry of protein interactions. In John R. Koza, David E. Goldberg, David B. Fogel and R. L. Riolo, editors, *Proceedings of Genetic Programming 1996*, pages 275–381, Cambridge, MA, 1996. MIT Press.
- [96] Michael L. Raymer, William F. Punch, Erik D. Goodman, Leslie A. Kuhn, and Anil K. Jain. Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(2):164–171, July 2000.
- [97] Michael L. Raymer, William F. Punch, Erik D. Goodman, P. C. Sanschagrin, and Leslie A. Kuhn. Simultaneous feature extraction and selection using a masking genetic algorithm. In T. Back, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA-97)*, pages 561–567, San Francisco, CA, 1997. Morgan Kaufmann Publishers.
- [98] Daniel Rivero, Juan R. Rabu nal, Julián Dorado, and Alejandro Pazos. Using genetic programming for character discrimination in damaged documents. In Guenther R. Raidl, Stefano Cagnoni, Jurgen Branke, David W. Corne, Rolf Drechsler, Yaochu Jin, Colin Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3005 of *LNCS*, pages 347–356, Coimbra, Portugal, 5-7 April 2004. Springer Verlag.

- [99] Daniel Rivero, Rafael Vidal, Julian Dorado, Juan R. Rabunal, and Alejandro Pazos. Restoration of old documents with genetic algorithms. In Günther R. Raidl, Stefano Cagnoni, Juan Jesús Romero Cardalda, David W. Corne, Jens Gottlieb, Agnès Guillot, Emma Hart, Colin G. Johnson, Elena Marchiori, Jean-Arcady Meyer, and Martin Middendorf, editors, *Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM*, volume 2611 of *LNCS*, pages 436–447, University of Essex, England, UK, 14-16 April 2003. Springer-Verlag.
- [100] Mark Roberts. *An Investigation into the use Genetic Programming for Image Compression*. BSc Dissertation, School of Computer Science, University of Birmingham, April 2000.
- [101] Mark E. Roberts. The effectiveness of cost based subtree caching mechanisms in typed genetic programming for image segmentation. In Günther R. Raidl, Stefano Cagnoni, Juan Jesús Romero Cardalda, David W. Corne, Jens Gottlieb, Agnès Guillot, Emma Hart, Colin G. Johnson, Elena Marchiori, Jean-Arcady Meyer, and Martin Middendorf, editors, *Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM*, volume 2611 of *LNCS*, pages 448–458, Berlin, 14-16 2003. Springer-Verlag.
- [102] Mark E. Roberts and Ela Claridge. Cooperative coevolution of image feature construction and object detection. In Xin Yao, Edmund Burke, Jose A. Lozano, Jim Smith, Juan J. Merelo-Guervós, John A. Bullinaria, Jonathan Rowe, Peter Tiño Ata Kabán, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 899–908, Berlin, 18-22 2004. Springer-Verlag.
- [103] Brian J. Ross, Frank Fueten, and Dmytro Y. Yashkir. Edge detection of petrographic images using genetic programming. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 658–665, San Francisco, USA, 10-12 July 2000. Morgan Kaufmann.

- [104] Paul Schrater. Lecture notes - feature extraction/selection. Available online from http://gandalf.psych.umn.edu/~schrater/schrater_lab/courses/PattRecog03/Lec22PattRec03.pdf, November 2003.
- [105] W. Siedlecki and J. Sklansky. A note on genetic algorithms for large-scale feature selection. *Pattern Recogn. Lett.*, 10(5):335–347, 1989.
- [106] Will Smart and Mengjie Zhang. Classification strategies for image classification in genetic programming. In *Proceedings of Image and Vision Computing Conference*, pages 402–407, 2003.
- [107] William Smart and Mengjie Zhang. Using genetic programming for multiclass classification by simultaneously solving component binary classification problems. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 227–239, Lausanne, Switzerland, 30 March - 1 April 2005. Springer.
- [108] Matthew G. Smith and Larry Bull. Feature construction and selection using genetic programming and a genetic algorithm. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 229–237, Essex, 14-16 April 2003. Springer-Verlag.
- [109] Matthew G. Smith and Larry Bull. GAP: Constructing and selecting features with evolutionary computation. In Ashish Ghosh and Lakhmi C. Jain, editors, *Evolutionary Computing in Data Mining*, volume 163 of *Studies in Fuzziness and Soft Computing*, chapter 3, pages 41–56. Springer, 2004.
- [110] Matthew G. Smith and Larry Bull. Using genetic programming for feature creation with a genetic algorithm feature selector. In Xin Yao, Edmund Burke, Jose A. Lozano, Jim Smith, Juan J. Merelo-Guervós, John A. Bullinaria, Jonathan Rowe, Peter Tiño Ata Kabán, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 1163–1171, Birmingham, UK, 18-22 September 2004. Springer-Verlag.

- [111] Stephen F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, 1980.
- [112] Stephen L. Smith, Stefan Leggett, and Andrew M. Tyrrell. An implicit context representation for evolving image processing filters. In Franz Rothlauf, Juergen Branke, Stefano Cagnoni, David W. Corne, Rolf Drechsler, Yaochu Jin, Penousal Machado, Elena Marchiori, Juan Romero, George D. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3449 of *LNCS*, pages 398–407, Lausanne, Switzerland, 30 March–1 April 2005. Springer Verlag.
- [113] John J. Szymanski, Phil C. Blain, Jeffrey J. Bloch, Christopher M. Brislawn, Steven P. Brumby, Maureen M. Cafferty, Mark E. Dunham, Janette R. Frigo, Maya Gokhale, Neal R. Harvey, Garrett Kenyon, Won-Ha Kim, J. Layne, Dominique D. Lavenier, Kevin P. McCabe, Melanie Mitchell, Kurt R. Moore, Simon J. Perkins, Reid B. Porter, S. Robinson, Alfonso Salazar, James P. Theiler, and Aaron C. Young. Advanced processing for high-bandwidth sensor systems, November 2000.
- [114] John J. Szymanski, Steven P. Brumby, Paul Pope, Damian Eads, Diana Esch-Mosher, Mark Galassi, Neal R. Harvey, Hersew D. W. McCulloch, Simon J. Perkins, Reid Porter, James Theiler, A. Cody Young, Jeffrey J. Bloch, and Nancy David. Feature extraction from multiple data sources using genetic programming. In Sylvia S. Shen and Paul E. Lewis, editors, *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery VIII*, volume 4725 of *SPIE*, pages 338–345, August 2002.
- [115] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303–309. Morgan Kaufmann, July 1994.
- [116] Astro Teller and Manuela Veloso. PADO: A new learning architecture for object recognition. In Katsushi Ikeuchi and Manuela Veloso, editors, *Symbolic Visual Learning*, pages 81–116. Oxford University Press, 1996.

- [117] James Theiler, Neal R. Harvey, Steven P. Brumby, John J. Szymanski, Steve Alferink, Simon J. Perkins, Reid B. Porter, and Jeffrey J. Bloch. Evolving retrieval algorithms with a genetic programming scheme. In Michael R. Descour and Sylvia S. Shen, editors, *Proc. SPIE Imaging Spectrometry V*, volume 3753, 2002.
- [118] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004.
- [119] R. Paul Wiegand, William C. Liles, and Kenneth A. De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In Günther R. Raidl, Stefano Cagnoni, Juan Jesús Romero Cardalda, David W. Corne, Jens Gottlieb, Agnès Guillot, Emma Hart, Colin G. Johnson, Elena Marchiori, Jean-Arcady Meyer, and Martin Middendorf, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, volume 2611 of *LNCS*, pages 1235–1245, Berlin, 14-16 2001. Morgan Kaufmann.
- [120] Jay F. Winkeler and B. S. Manjunath. Genetic programming for object detection. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 330–335, San Francisco, CA, USA, 13-16 1997. Morgan Kaufmann.
- [121] Jay F. Winkeler and B. S. Manjunath. Genetic programming for object detection. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 330–335, San Francisco, CA, USA, 13-16 1997. Morgan Kaufmann.
- [122] Hau-San Wong and Ling Guan. Application of evolutionary programming to adaptive regularization in image restoration. *IEEE Transactions on Evolutionary Computation*, 4(4):309–326, November 2000.
- [123] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Trans. Evolutionary Computation*, 3(2):82–102, 1999.

- [124] Mengjie Zhang. *A Domain Independent Approach to 2d Object Detection Based on Neural Networks and Genetic Paradigms*. PhD thesis, Department of Computer Science, RMIT University, Melbourne, Victoria, Australia, August 2000.
- [125] Mengjie Zhang and Urvesh Bhowan. Program size and pixel statistics in genetic programming for object detection. In Guenther R. Raidl, Stefano Cagnoni, Jurgen Branke, David W. Corne, Rolf Drechsler, Yaochu Jin, Colin Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3005 of *LNCS*, pages 377–386, Berlin, 5-7 2004. Springer Verlag.
- [126] Mengjie Zhang, Victor B. Ciesielski, and Peter Andreae. A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Applied Signal Processing*, 2003(8):841–859, July 2003. Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis.
- [127] Mengjie Zhang and Will Smart. Multiclass object classification using genetic programming. In Guenther R. Raidl, Stefano Cagnoni, Jurgen Branke, David W. Corne, Rolf Drechsler, Yaochu Jin, Colin Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3005 of *LNCS*, pages 369–378, Coimbra, Portugal, 5-7 April 2004. Springer Verlag.