The University of Birmingham
School of Computer Science
MSc in Advanced Computer Science

Project Report

Detecting Faces using Evolution

John Montgomery
msc37jxm@cs.bham.ac.uk
Supervisor: Dr J.L. Wyatt

September 7, 2004

Abstract

Face detection is the process of locating faces in images - a task which humans seem to have no difficulty with, compared to computers. There are many potential applications for face detection in the field of computing and would be a key ability for socially interacting robots.

In this project, evolution is used to train several "ratio-templates" as face/non-face classifiers. Combinations of separately evolved "ratio-templates" form the basis of a face detector that is capable of detecting upright frontal views of faces.

The system is successfully demonstrated, on several images containg multiple faces and using a web-cam, with much promise. In particular a sufficiently low false-positive rate and a respectable true-positive rate are apparent.

Keywords

Evolution, Ratio-Templates, Face Detection, Computer Vision.

Contents

1 Introduction **5**
1.1 Face Detection . . . 5
1.1.1 Handling Multiple Scales . . . 5
1.1.2 Pattern Recognition . . . 7
1.1.3 Training Classifiers . . . 8
1.1.4 Ratio-Templates . . . 9
1.1.5 Evolution for Computer Vision . . . 10
1.2 Terminology . . . 10

2 Implementation **12**
2.1 Software Development . . . 12
2.1.1 Division of Labour . . . 12
2.1.2 OpenCV . . . 13
2.2 Faces for Training . . . 13
2.3 Evolving Ratio-Templates . . . 13
2.3.1 Genetic Algorithm . . . 16
2.4 Genotypes . . . 16
2.4.1 RTCandidate . . . 17
2.4.2 MRTCandidate . . . 17
2.4.3 NEATCandidate . . . 18
2.5 Boot-strapping Non-Face Images . . . 18
2.6 Face Detection on Arbitrary Images . . . 20
2.6.1 Building a Better Detector . . . 20
2.6.2 Hillclimbing . . . 22
2.6.3 Heuristic Tidying . . . 23

3 Experiments and Results **25**
3.1 Data Sets . . . 25
3.2 Experiments . . . 27
3.2.1 ROC Graphs . . . 27
3.2.2 Experimental Results . . . 27
3.3 Improving the Results . . . 34
3.3.1 AND'ing . . . 34
3.3.2 OR'ing . . . 35
3.3.3 Logical Grouping . . . 36
3.3.4 Detecting Faces in (not quite) Real-time . . . 37
3.3.5 Hillclimbing . . . 37

4 Extensions, Improvements and Conclusion **44**
4.1 "Better" Genotypes 44
4.1.1 Enforce Symmetry 44
4.1.2 Enforce Neighbourliness 44
4.2 Better, Faster, Stronger Classifiers 45
4.2.1 Cascading Ratios 45
4.2.2 Combining Ratio-Templates 45
4.3 Handle Rotation 46
4.4 Conclusion 46

A Electronic Resources **49**

List of Figures

1.1 Using an Image Pyramid for Scale Invariant Detection 6

2.1 Face Alignment 14
2.2 Matching Ratio-Templates to Images 15
2.3 Steady-State Genetic Algorithm 16
2.4 Boot-Strapping Non-Face Images 19
2.5 ANDing to reduce false-positives 21
2.6 Grouping of Templates for Detection 22
2.7 Merging Overlapping Detections 23

3.1 Face Images 26
3.2 ROC Graph - MRT Lenient Sinha 30
3.3 ROC Graph - MRT Lenient Rowley 30
3.4 ROC Graph - MRT Lenient Random 31
3.5 ROC Graph - MRT Strict Sinha 31
3.6 ROC Graph - MRT Strict Rowley 32
3.7 ROC Graph - MRT Strict Random 32
3.8 ROC Graph - MRT Moderate Sinha 33
3.9 ROC Graph - MRT Moderate Rowley 33
3.10 ROC Graph - MRT Moderate Random 34
3.11 Training Data Detections, using simple combination of templates 38
3.12 Training Data Detections, using simple combination of templates (continued) 39
3.13 Training Data Detections, using simple combination of templates (continued) 40
3.14 Training Data Detections, using simple combination of templates (continued) 41
3.15 Testing on Group Photos 42
3.16 The Author's Face being Detected 43

List of Tables

3.1 Average Precision and Recall 28
3.2 AND'ed Precision and Recall 35
3.3 OR'd Precision and Recall 36

Chapter 1

Introduction

Face detection is the process of locating faces in images - a task which humans seem to have no difficulty with, compared to computers. This immediately marks it out as an interesting area in Artificial Intelligence, where we often want to make computers better at the things humans take for granted.

Face detection is related to face recognition, but is in some respects harder. Face recognition is usually concerned with finding the "best match" to a known face and so methods like Principle Component Analysis (PCA) are are often used[7].

Face detection is however often a part of a face recognition system. The system needs to know if a face is present in an image and then where it lies, so that the face can be extracted and offered to the recognition module.

Other uses for face detection include hands free user interfaces[3] and eye finding[23][1].

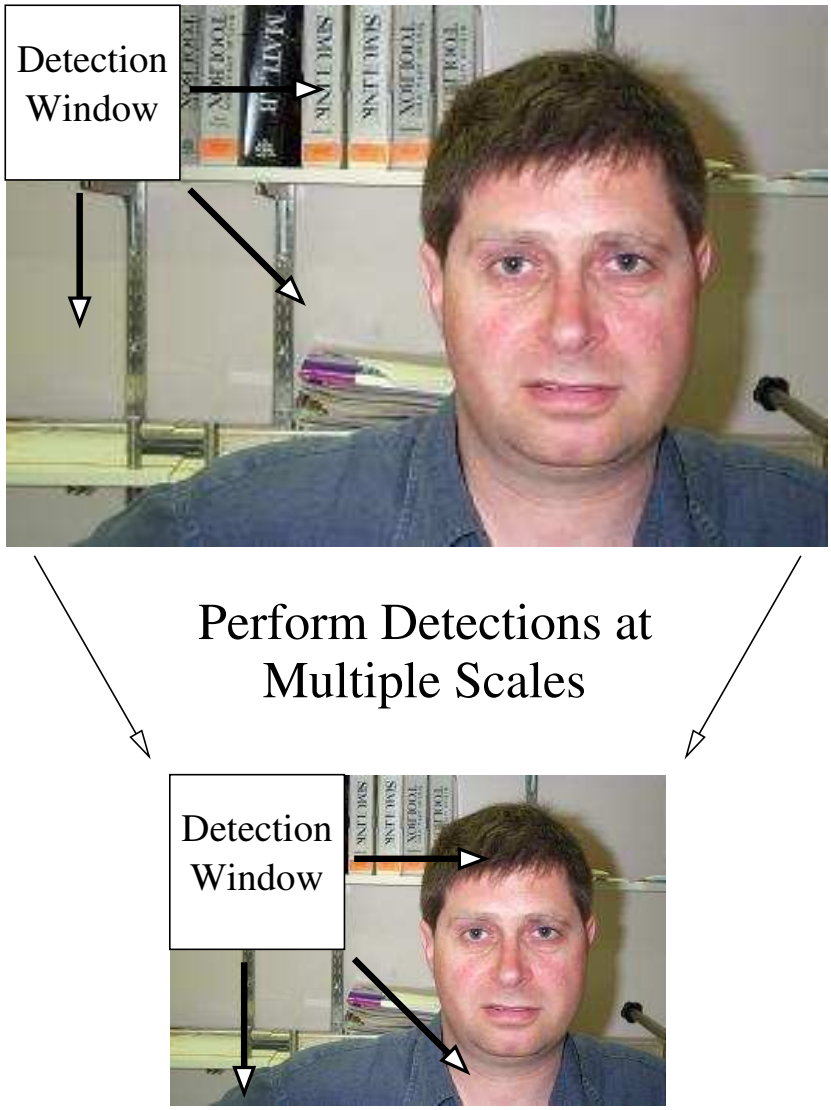
For a face detection system to be useful it needs to be quick, accurate, able to handle faces at varying distances[2] and able to cope with differing light conditions.

1.1 Face Detection

1.1.1 Handling Multiple Scales

A standard approach to face detection is to use a classifier and an image pyramid to allow the detection of faces at different scales[11][24][20][21]. An image pyramid is formed by repeatedly down-scaling[3] (shrinking) an image to form a virtual "pyramid" of images. One can then use a classifier to classify fixed-size "detection windows" at every position, of each image, in the pyramid. The particular image in the pyramid that a detection occurs within dictates the "scale" of the detection - the smaller the image the larger the scale. Combining the scale with the position within the (scaled) image, that the detection occurs, makes it possible to calculate the size and position of the detection relative to the original image. This process can be seen in Figure 1.1.

[1] useful for discovering eye contact
[2] yielding faces at various scales within an image.
[3] e.g. via bilinear interpolation



1. Scan a 20x20 pixel "detection window" over every position in the image.
2. Use a detector to classify each detection window image (noting any matches and compensating for the current scale).
3. Shrink the image slightly (x 0.8 roughly) and record the new scale.
4. If the image is still larger than the detection window goto 1.

Figure 1.1: Using an Image Pyramid for Scale Invariant Detection

One must be careful when using an image pyramid, to ensure that it is created properly. The relative sizes between images in the pyramid will have a big effect on the pyramids usefulness for detection. If a face was to lie *in between* two image scales then this would present a failure of the pyramid. Typically the scale factor is about 1.2[20], between the smaller and the larger images, i.e. the smaller image is about 0.8 times the size of the larger.

One way to speed up detection using an image pyramid is to not check every position and/or scale. If we are system is being used on video one can consider motion as a useful indicator of where to look - as faces are more likely to be moving than the background. Scassellati[23] used a "motion-based pre-filter" that only considers positions that have either recently contained a face, have recently seen motion or else have not not been considered in the last three seconds. This means that faces that appear suddenly will tend to get noticed and faces previously detected will tend to carry on being detected.

Rather than using an image pyramid to rescale the source image, one also can instead rescale the "detection window"[27]. One takes the source image and slides detection windows of multiple scales over the image at every position. This is in some ways quite similar to the image pyramid, but relies on the classifier being able to handle images of different sizes. It is possibly more time consuming, as the classifier then cannot be optimised for a fixed image size. It may however allow more accurate detections to occur with large faces, as more information is presented to the classifier. However the net effect is more or less the same as using an image pyramid.

By contract, the work of Viola and Jones[26] *does not* make use of an image pyramid. Instead they make use of an "integral image" type, which allows the very rapid evaluation, of the simple feature detectors they use, at multiple scales. This integral image, along with the "cascading" of feature detectors, is cited as part how they were able to make use of 200 feature detectors and still maintaining a good performance - speeds in the region of 15 frames per second on a "conventional desktop"[26]. Apparently this is about the same speed that it takes to compute a "12 level image pyramid alone"[26], so this is a fairly significant speed advantage.

1.1.2 Pattern Recognition

If we are using a system (such as an image pyramid) to provide a classifier with images to classify, we have effectively reduced face detection to a pattern recognition problem. All that is then needed is something that can recognise faces.

Neural Networks would be one obvious choice for pattern recognition and many systems do indeed make use of them in face detection systems[11][20][21][27]. The majority of "neural computing" applications after all "are concerned with problems in pattern recognition"[5]. Neural Networks whilst powerful do require tuning for use in face detection. The choice of network architecture can have a large effect on the systems performance. Wiegand, Igel and Handmann[27] have demonstrated the use of evolution to optimise neural network architecture for system speed, whilst still maintaining accuracy.

Kirchberg, Jerosky and Frishholz[13] use a Hausdorff Distance-Based model for classification. They use evolution to optimise the model. They concentrated on finding only a single face in an image, so they do not strictly speaking use

their model as a classifier, picking instead the best match to the model in an image. However "the extension to finding multiple faces is straightforward"[13] and would probably involve merely applying a threshold to the distance indicated by the Hausdorff Distance metric. In this way it could function as a straightforward face/non-face classifier.

Ratio-Template methods (see Section 1.1.4) usually rely on careful hand-crafting[24][23][1]. Although Sinha does discuss[24] a possible system for "learning the invariant" properties needed by a ratio-template the learning system is only demonstrated using very basic synthetic images. This marks them out from a lot of other methods which normally have some element of training. As of such they make good candidates for further investigation.

An Alternative to the Holistic Approach

Holistic methods[4] are probably the most common types. It feels quite natural to treat the face as a single object to be detected, with the individual elements serving as distinguishing features of the whole, rather than separate elements. It would seem *unusual* not to recognise faces as a whole[22]. One draw back of this approach is that the obscuring of the face may result in the face being undetected, but depending on the application domain this may not be a problem.

However even with holistic methods it is often the intention to locate facial components[23]. By finding the facial components we have more information, which may prove useful. In fact this extra information has been used by some to further improve upon the tracking ability of otherwise holistic methods[3].

By using a combination of neural network feature detection and fuzzy logic Kouzani[14] has shown a non-holistic method that is able to cope with the occlusions of some facial features. The system is also able to deal with pose estimation, as knowledge of the facial feature positions allows inference of the position and orientation of the face. However Kouzani's system does (at least for the current implementation) appear to be rather slow - "tested on a Sun-Spark 20 station ... typical 512 x 384 image is about 9.3 min". The fact that a non-holistic method might be slower than holistic ones is hardly surprising, as they typically involve much more work - requiring first the detection of multiple facial elements and then the inference of face locations.

1.1.3 Training Classifiers

In order to create a face-detection system a classifier of some sort is required. This classifier simply has the task of discriminating between face and non-face images. Whilst it is possible to devise such classifiers manually, it is often the case that we would prefer to automate this task.

To train such classifiers we will typically need examples of both face and non-face images. The fidelity of both of these sets will obviously have a large impact on how effective the training is.

Positive Examples

Creating example image of faces for training takes two forms:

[4] ones that recognise the face as a whole

- Synthetic images.
- Real images (possibly transformed).

Using 3D head models[14] it is possible to generate synthetic images of faces. This has the advantage of making the generation of faces quite automatic, but does require the creation (or acquisition) of the 3D models and careful use. However if care is not taken, we might inadvertently introduce a bias to the data, resulting in a bias of the trained classifier.

Alternatively we can take real images containing faces and extract the faces regions. This usually involves a certain amount of manual intervention, by marking out the face or its features[20][21]. We can then extract the face, rotating and/or rescaling it as needed. Rotation and rescaling will usually be required, so that the final face image will be in a format suitable for presenting to the classifier. e.g. containing an upright face.

Negative Examples

Kirchberg, Jesorsky and Frischholz[13] do not use negative (non-face) examples for training their system. They are only able to do this as they have a metric (Hausdorff Distance[13]) for discovering how similar their edge models are to the face images used in training.

With other approaches the only feedback available is how well the classifier can discriminate between face and non-face examples. It is just as important (if not more important) to have a good system for generating non-face images. Whilst it is quite easy to generate a set of faces that one could consider representative of most faces, it is very difficult to create a "representative" set of non-face images[20]. This is because the set of non-face images consists of *everything* that is *not* a face. Face images are easy to characterise and thus represent, non-face images are not.

One effective technique to deal with this is to "boot-strap" the non-face images[20]. This involves adding new non-face images as training progresses, based on the current performance of the system. This process is outlined, as applied to the author's system, in Section 2.5.

1.1.4 Ratio-Templates

Ratio Templates are a form of "image invariant" proposed by Sinha[24]. Sinha observed that under normal[5] lighting there were often "invariant" relationships between the brightness of various regions of the face. The most striking of these is apparent with the eye regions, in that they are almost always darker than the rest of the face. So an image containing two dark regions close to each other has a better chance of being a face than an image without such regions. As the relationships are between the *relative* brightness in different regions, this lends a certain robustness to the technique. Overall image brightness will therefore tend to have little effect on the detection process.

Sinha initially used a very simple ratio-template that was designed from looking at real faces. Scassellati[23] further improved Sinha's ratio-templates and it as viable for real-time detection of faces in an active vision system. By

[5] i.e. lit from the top or sides, not below

altering Scassellati's hand-crafted ratio-template, to conform to the proportions of the "golden ratio", Anderson and McOwan[1] were further able to improve tolerance to illumination changes. This clearly demonstrates that given Sinha's basic scheme there was much improvement that could be made. As of such it seems to indicate that it may be possible to evolve ratio-templates, as the differences between the templates used in the three cases are incremental.

The exact use of ratio-templates is outlined in Section 2.3, as part of the implementation of the final system.

1.1.5 Evolution for Computer Vision

Artificial Evolution has long been used for "optimisation" of system parameters. Obvious uses within the field of computer vision entail evolving the various parameters used by vision sub-systems (thresholds, aperture sizes etc), which would otherwise traditionally have to be tuned by hand[17]. Evolution can also be applied in a "holistic" manner[18] to tune "low, medium and high level processing" parameters in concert, leading to potentially better global solutions than possible by individually optimising each separate sub-system.

Genetic Programming (GP) has great appeal for computer vision. There is something quite visceral about evolving programs. Genetic Programming has been applied to detecting targets in SAR[6] Imagery[10], multi-class object detection[28][28] and Object Tracking[19] amongst others. Compared to *basic* parameter optimisation, Genetic Programming has a large scope for evolving interesting and novel solutions. However GP's performance often depends heavily on the initial configuration. Whilst this is usually true for most evolutionary approaches, GP seems particularly sensitive to this. The choice of function and terminal sets is crucial, at the very least there must be "closure"[15] so that it is *possible* to generate a correct solution, but one must also consider as few terminals and functions as possible if evolution is to progress at a reasonable rate. In short GP can only work with the "tools" it has been given. If these tools are not up to the job, or combining of these tools is difficult then GP will not perform very well.

Somewhere in the middle of these two extremes we have other evolutionary approaches, such as using evolution to create neural networks[27]. In general it would seem wise to use an evolutionary approach that is as complex as needed, but no more so. If we start off with something too complex (e.g. GP) we never get good results, as getting everything setup right may take too long.

1.2 Terminology

In order to avoid ambiguity some terminology needs defining. Most of these terms are commonly used to describe classification performance[8]. The most basic terms (when applied to face/non-face classification) are:

- Positives (P) - the number of face images.
- Negatives (N) - the number of non-face images.

[6] Synthetic Aperture Radar

- True-Positives (TP) - the number of face images that have been correctly classified.
- False-Positives (FP) - the number of non-face image that have been incorrectly classified.
- True-Negatives (TN) - the number of non-face image that have been correctly classified.
- False-Negatives (FN) - the number of face image that have been incorrectly classified.

Most of the other terms used here rely on these basic concepts. Particularly in cases where the size of the sets being classified is "skewed"[8], the basic terms may not always be as useful. So such extra terminology is very handy.

- true-positive rate $= TP/P$
- false-positive rate $= FP/N$
- accuracy $= (TP + TN)/(P + N)$
- precision $= TP/(TP + FP)$
- recall $= TP/P$

The obvious point to mention with these terms is that "recall" and "true-positive rate" are one and the same, but it is sometimes nicer to talk about recall rather than the true-positive rate.

As there are typically only a small number of face images compared to non-face images, accuracy is not always a useful measure either. If we had 500 face images and 10,000 non-face images, then we could achieve and accuracy of 0.95, by simply always classifying images as non-faces. However accuracy has been defined here, so as to help avoid any confusion in its use.

Precision is quite a useful term though, as it gives us an indication of how likely a positive classification is to be accurate. If we have a high precision classifier, then we can be pretty certain that anything it classifies as a face image is very likely to be a face image. However one could also have a high precision classifier that only hardly ever classifies anything as a face. The precision of a classifier is usually accompanied by it's recall, to better give an idea of the classifier's overall behaviour.

Chapter 2

Implementation

The system presented here is for the detection of "upright, frontal views of faces in gray-scale images"[20]. "Ratio-Templates"[24] are evolved, to form classifiers capable of discriminating between fixed sized images of faces and "non-faces". Evolution is performed using a steady-state genetic algorithm, coupled with a "boot-strapping" phase. The boot-strapping phase helps us to efficiently produce "non-face" images as counter-examples during evolution.

The evolved ratio-templates/classifiers are grouped together to improve their accuracy and precision. The combined classifiers are used as a (crucial) component in the face-detection system, capable of detecting face positions and approximate sizes in arbitrary images.

Apart from the creation of the data sets and some of the initial configurations of the ratio-templates, there is actually nothing inherently biased towards detecting faces in the system described. In principle given different examples of positive and negative images one could evolve a detector to locate other objects. By increasing the specialisation of the system we could probably reap large benefits for this particular task, but the current performance was found to be adequate. The generality would also be useful for extending the system to detect faces that are not "upright, frontal views"[20].

2.1 Software Development

The software was developed using a combination of Python[16] and C++. SWIG (Simple Wrapper and Interface Generator)[4] was used to generate the "glue-code" to allow Python code to use of custom written C++ extensions.

2.1.1 Division of Labour

The software is roughly divided into two parts. One part is concerned with evolving ratio-templates that can distinguish between face and non-face images. This part is mostly written in Python, as it does not need to run in real-time, but makes use of some C++ classes, particularly during fitness evaluation. The other part is mainly written in C++ and utilises (potentially) several ratio-templates to detect faces within an image at various scales and positions. This necessitates a large number of evaluations of the ratio-templates and so C++

was used to ensure this happens at a reasonable rate, even on modest hardware[1]. Having said that speed was never the main aim of this project and as of such, much could still be done to improve this aspect.

In addition, a simple GUI application was written in java, to aid in the preparation of the face images needed for training.

2.1.2 OpenCV

Intel's OpenCV [12] library was used to handle images, due in part to the author's familiarity with the library. The OpenCV library provides a very rich and deep level of functionality, which was only barely touched upon by this project. However having easy access to some highly optimised computer vision algorithms seemed like a very good idea - just in case.

2.2 Faces for Training

In order to train the templates to classify face and non-face images we need to have some examples. Example images of human faces viewed from the front were obtained from the Caltech Computation Vision Archive [2] and AT&T [6]. These images although featuring faces viewed from the front still needed adapting for use in the training process. As we are (only) trying to detect upright faces it is important that we have an appropriate data set. People looking into a camera do not always have an absolutely upright stance and we really need control over this if we want to make training possible. So following Rowley et al [20] the positions of six features (eyes, nose-tip, corners and middle of mouth) were manually tagged. These positions were then used to translate, rotate and scale the source images so that they best fit a 20x20 image suitable for training. This process was done in a relatively straight-forward "brute-force" manner (see Figure 2.1).

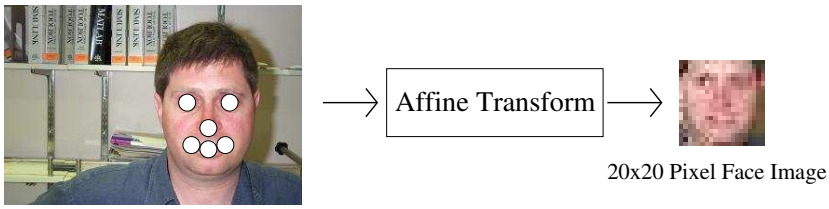
The fact that the alignment process is automated, requiring only manual intervention at the tagging stage, means that we can easily generate slightly varied data sets. For example, we could introduce a slight fluctuation in the orientation or scale of the faces to make the data noisier. We could also produce data sets with the faces rotated towards a different (known) angle, which would be useful if we wanted to train for detection of faces at other/multiple angles.

2.3 Evolving Ratio-Templates

The face detection method chosen to investigate was that of Sinhas Ratio-Templates[24]. Normally these are hand-crafted[24][23][1], so it was decided to see whether it was possible to evolved their configurations instead. The ratio-template approach is quite appealing as it gives us a nice visual model to look at and also provides a simple and fast detector. By comparison with the neural networks used by Rowley et al.[20] there are far fewer computational elements involved in a ratio-template and they are far less opaque to examine.

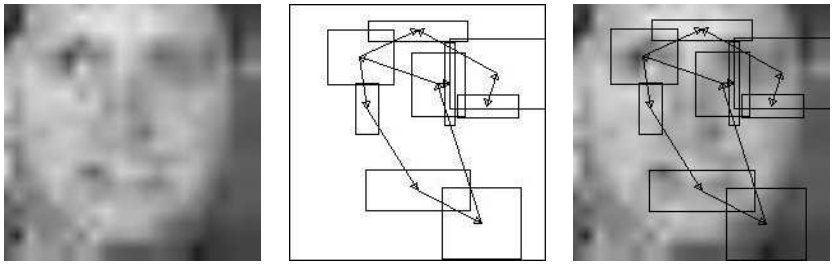
A ratio-template consists of a series of "regions" and "ratios" between pairs of regions. Regions are used to sample the brightness in an image and the ratios

[1] seconds rather than minutes for a 160x120 image on a P3 500Mhz laptop



1. Calculate the translation of the feature coordinates so that their average position lies at the origin.
2. Try all half-degree (720) rotations and find which one minimises the distance between the coordinates and an "ideal" set of coordinates.
3. Try all scales from 1/720 to 1.0 and find the one which best minimises the distance from the ideal coordinates (we assume the real face needs shrinking rather than enlarging).
4. Calculate the translation of the coordinates so that they lie inside the 20x20 image correctly (rather than being centred at the origin).
5. Combine the the translations, rotation and scaling to form an Affine transform suitable for converting the original image into a 20x20 image containing just the face.

Figure 2.1: Face Alignment



1. Look at each ratio in turn:
   (a) If $brightness(region_1)/brightness(region_2) > threshold$ then the ratio is "present" in the image.
   (b) If $brightness(region_1)/brightness(region_2) <= threshold$ then the ratio is "absent" from the image.
2. Sum the weights of the "present" ratios and subtract the weights of the "absent" ratios, comparing the result with zero:

$$\left(\sum_{i}^{present} weight_i\right) - \left(\sum_{j}^{absent} weight_j\right) > 0 \tag{2.1}$$

3. A result greater than zero indicates a match.

Figure 2.2: Matching Ratio-Templates to Images

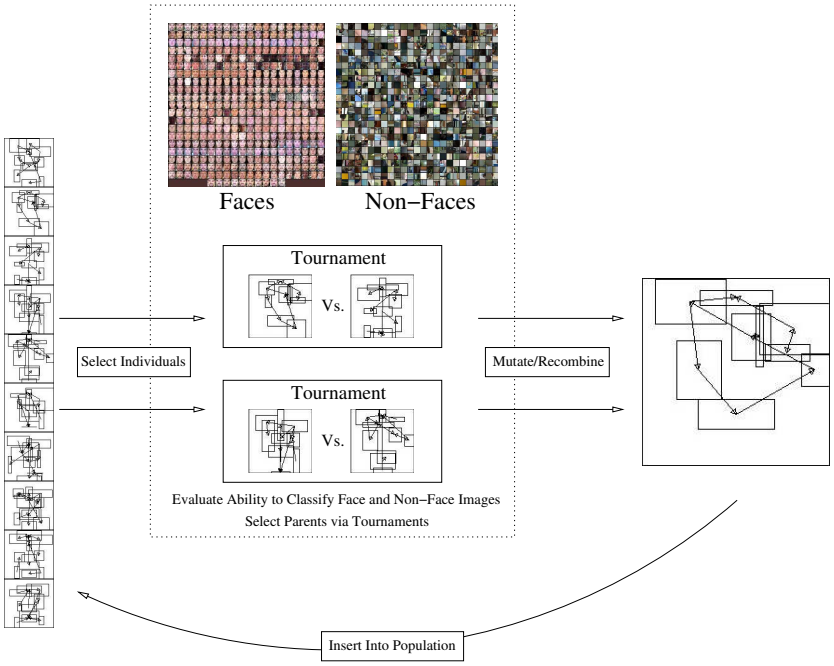
are used to make comparisons between the brightness of regions. We see how many of the ratios are present/absent within an image. As some ratios may be more important than others we also weight the ratios to help decide if a match has occurred (see Figure 2.2).

In this way the absence of a ratio can have just as much of an effect on the outcome as its presence. For example, a ratio that corresponding to the eye region versus the forehead might be very important and its absence could often indicate that the image being examined does not contain a face.

For optimisation purposes there are several attributes that spring to mind:

- Number of regions.
- Number of ratios.
- Position and size of regions.
- Ratio positions (i.e. which regions they apply to).
- Ratio thresholds and weights.

Exactly how we evolve these attributes will obviously depend on the genotypic representation used.



1. Select parent(s) via tournament selection, based on ability to classify face and non-face images.
2. Recombine and/or mutate parent(s) to produce an offspring.
3. Select a random population member and compare with the offspring.
4. If the offspring is better (or with a small probability) replace the population member with the offspring.

Figure 2.3: Steady-State Genetic Algorithm

2.3.1 Genetic Algorithm

A fairly standard steady-state genetic algorithm was used to evolve the ratio-templates (see Figure 2.3). At each step of the algorithm parents are selected via binary tournaments and used to create a single offspring, via a combination of mutation and/or recombination[2]. This offspring is then evaluated and replaces a random member of the population if the offspring is an improvement over that population member or with a small probability otherwise.

2.4 Genotypes

Three different genotypes were created, each slightly more complex than the last. In all cases there is a genotype to phenotype mapping, although this is not a very

[2] with certain probabilities

complex mapping, it is worth acknowledging. The genotypes are concerned with mutation and recombination, whereas the phenotype (the actual ratio-template object) is purely concerned with ratios, regions and matching/classifying images. The major advantage of this mapping is that the final detector need only be concerned with the phenotypes (ratio-templates), leaving us free to potentially create quite arbitrary genotypes. This means we can decouple the evolutionary process from the final face detection process.

In all cases, the genotypes regions are initialised in a similar fashion, either using a hand-crafted or randomised layout. This is necessary for the RTCandidate in particular, but also helps the other genotypes get a reasonably good starting point. An obvious specialisation would be to enforce symmetry[13] in the final phenotypes, which would have the benefit of reducing the potential search space, but this was not attempted at this stage.

To help encourage ratios to form between neighbouring regions mutation, in all of the candidates, mutation was biased towards selecting nearer regions. This bias was very mild and used a simple ranking to preferentially select the nearer regions to the region currently used in a ratio.

2.4.1 RTCandidate

The RTCandidate allows the optimisation of a fixed number of ratios. The genotypic representation of a ratio mirrors that found in a ratio-template. Each ratio has a threshold, weight and a pair of indices representing the regions that the ratio applies to. In addition there is also a used/un-used boolean, which signifies whether we shall use the ratio in the final phenotype or not. This allows for a certain amount of variablility in the size of the phenotype and allows us to maintain the simplicity of a fixed-length genotype.

The genotype is implemented as a series of flat arrays, with the values of a single ratio being spread across multiple arrays - one for each ratio attribute (used, threshold, weight and the region indices). This was done so that we have several arrays of homogenous values (booleans, integers and real numbers), which makes mutation and recombination easier to implement. This also means that we have two classic genotypic representations, one being discrete and one continuous. Mutation follows this pattern. The discrete section of the genotype is subject to a point-rate mutation, averaging one alteration per mutation. The continuous section is mutated by adding a small amount of Gaussian noise to every element in the array, as is classically done when optimising real-valued parameters. Recombination is handled via discrete uniform recombination, at the ratio level. Each ratio in the offspring is drawn in its entirety from one or other parent. This makes sense as the ratios form obvious "building blocks"[9].

The major drawback of this genotype is that if the regions (manually) specified for use in the template are not appropriate for the task at hand, then little can be done about this.

2.4.2 MRTCandidate

The MRTCandidate extends the RTCandidate, so it too allows the optimisation of a fixed number of ratios. In addition it allows the optimisation of the positions and sizes of a fixed number of regions. The x,y location and sizes are represented as real-valued numbers between 0 and 1. In the genotype the x,y location

represents the centre point of a region, so that increases or decreases in size will have an effect in all directions, rather than arbitrarily only changing the regions coverage to the right and/or left.

Mutation of the regions is only performed with a small probability[3] - most of the time only the ratios are mutated. This allows the regions to shift slightly in position, but gives enough time for the ratios to be optimised to match them. Recombination of the regions is performed in addition to that of the the ratios and is performed in a similar fashion, with regions being transferred whole from either parent to the offspring with equal probability. One slight drawback of this approach is that it may result in disruption of the ratios, as the regions that they refer to *may* alter radically after recombination. However this is only a problem when both parents are massively different, in which case we would not expect recombination to be useful anyway.

This genotype addresses the drawback of the RTCandidate, allowing candidates to adjust the regions to better fit the data/objects being modelled.

2.4.3 NEATCandidate

This final genotype was inspired by NEAT (NeuroEvolution of Augmenting Topologies) [25]. The intent was to create a more dynamic genotype that can increase in complexity over time.

Traditionally creating recombination operators for dynamic genotypes is quite awkward. NEAT assigns "unique historical tags" to each newly added node/connection in a network and these are used to "line-up" the two genotypes for recombination. This means that we will combine features that were originally the same, but may have been altered by mutation since. Thus the offspring will typically contain the features that are common in both parents, something which might otherwise be quite tricky to do when there is no explicit indication of commonality.

2.5 Boot-strapping Non-Face Images

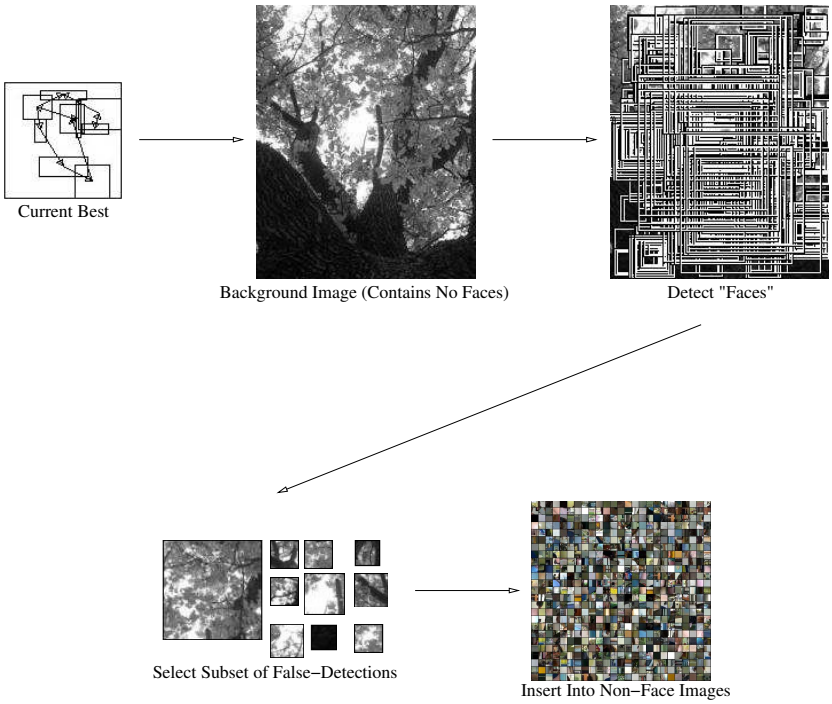
The "boot-strapping" method employed by Rowley et al[20] was adapted for use with the steady-state genetic algorithm. This technique was necessary to avoid a "huge training set for non-faces"[20], whilst still maintaining a representative sample of non-face images. The boot-strapping process is only used periodically[4] and is outlined in Figure 2.4.

The boot-strapping has the effect of forcing the search to tackle areas the population's best does not currently handle very well, driving evolution towards "the precise boundary between face and non-face images"[20]. The immediate effect may well mean the "best" is suddenly no longer the best and this is part of the intent, as it helps prevent stagnation.

Another subtle benefit of the boot-strapping process is that every evolutionary run will result in the use of different non-face sets. This means that the final output of each run will produce candidates that are better able to handle different sets of non-face images. By combining the outputs of multiple

[3] e.g. 0.01

[4] say once every ten generations (or equivalent for a steady-state GA)



1. Select the current "best" candidate.
2. Select a background image, known to contain no faces, at random.
3. Use the candidate to match faces in the background image (see Section 2.6).
4. Insert a subset of the sub-images representing any (implicitly) false detections into the non-face set.

Figure 2.4: Boot-Strapping Non-Face Images

ratio-templates we can take advantage of these differences to further enhance detection performance.

2.6 Face Detection on Arbitrary Images

The GA described previously does not in itself evolve templates for detecting face positions in arbitrary images. It merely evolves templates that are (hopefully) able to discriminate between "face" and "non-face" images of fixed sizes.

Rowley et al. [20] and Sinha [24] both make use of "image pyramids" to handle detections of faces at different scales. To do this we simply try out our detector at every position in an image, noting any positions that match. We then scale the image down and repeat the process. Detections that occur on the smaller images represent larger faces and vice-versa (see Figure 1.1).

2.6.1 Building a Better Detector

This represents a lot of images to be viewed and so even with reasonable false-detection rates under training, we are probabilistically still quite likely to have several false-positives occurring. For example to detect faces in a $160*120$ image would involve examining roughly 37000 sub-images, of which we would expect only a few to contain faces. So even with a false-positive rate of only 0.001 we might still expect to have 30 or so false-positives. If our image only contains one face this would be quite disastrous, we might never know which detection was the real face. In fact a false-positive rate closer to 1 in 1,000,000 is probably needed "for a face detector to be practical for real applications"[26].

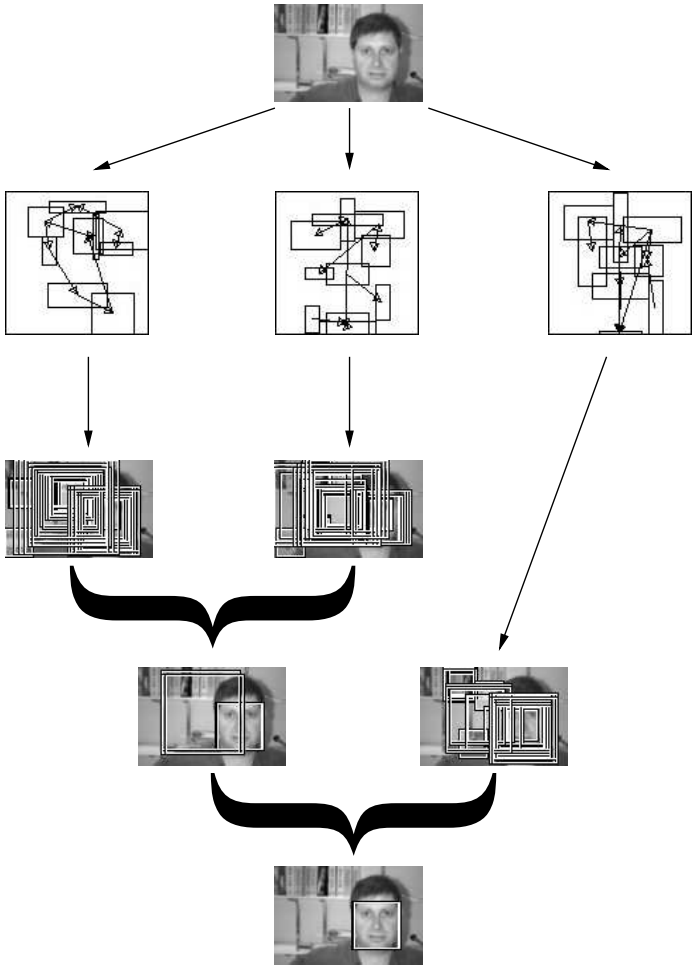
AND'ing

To combat false-positives we can combine several independently evolved templates and arbitrate between them. The most straightforward arbitration scheme is to simply AND the results [21], i.e. every template must agree with a detection otherwise we do not consider it a true detection (see Figure 2.5). This works because the "false-positives" that the templates give tend to differ, as each template has been evolved separately. Whereas the templates will tend to agree on the detection of the actual faces. This scheme works best when the templates being combined tend towards false-positives rather than true-negatives, i.e. precision is bad, but recall is good. If a single template is unable to detect a particular face, then this scheme will mean that the face is not detected (even if other templates being used could detect the face).

Another nice side benefit of using AND'ing to combine the templates is that it is not a costly process. When checking a specific scale and position we can "short-circuit" the logic (cascade the detectors[26]) and so may not need to use every template before we have decided that there is no face present.

Logical Grouping

Once we have a series of ratio-templates that result in low false-positive rates we can also consider logical grouping. This is quite a simple idea and just involves taking the resulting of several AND'ed groups of templates and OR'ing together those results (see Figure 2.6). This can be done to raise the total true-positive



By only accepting detections that all three templates agree upon, we have been able to get an accurate detection.

Figure 2.5: ANDing to reduce false-positives

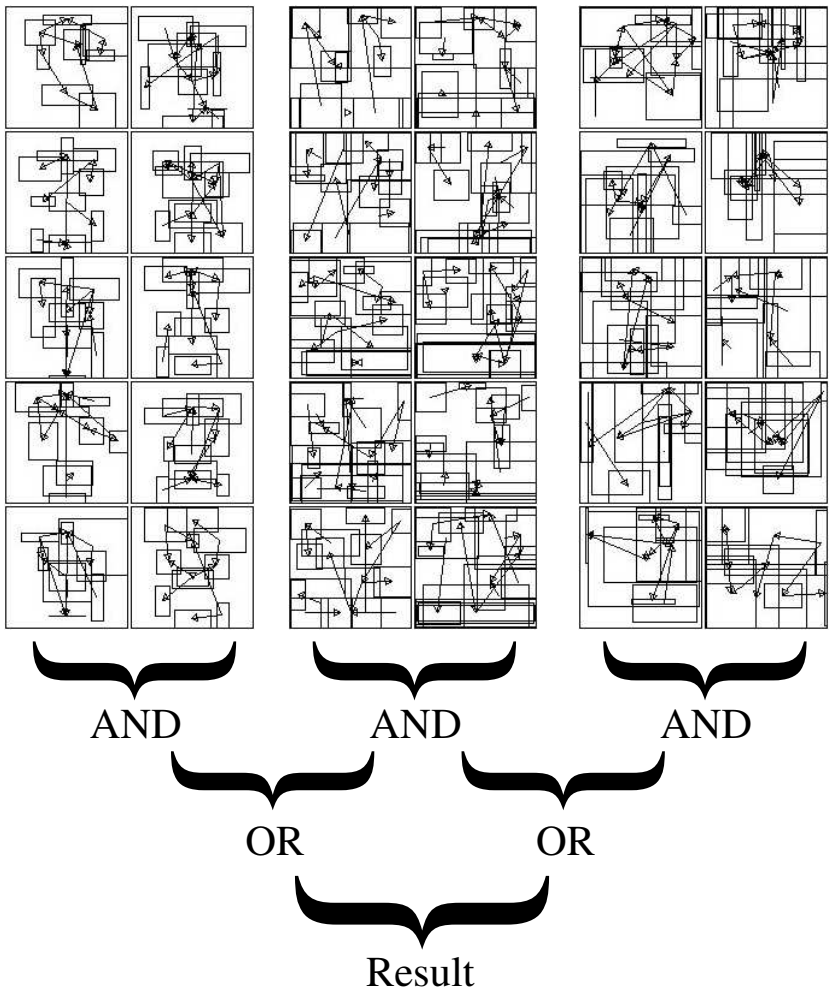


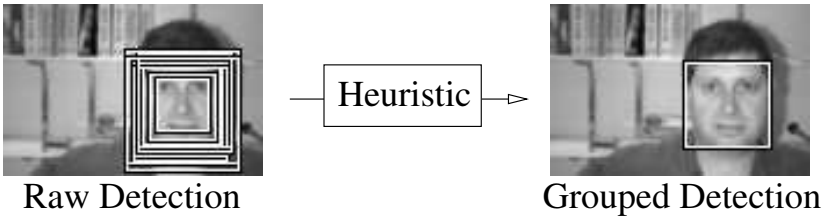
Figure 2.6: Grouping of Templates for Detection

rate, whilst still maintaining a lower false-positive rate. It also means that we could combine groups of detectors to detect faces at slightly different angles and by combining the results gain a detector capable of detecting faces at a variety of angles.

2.6.2 Hillclimbing

As the task of hand-selecting even a few templates to combine into a detector can be quite time consuming, it was decided that this process could do with being automated. As an experiment, a simple mutation-based hill-climbing algorithm was tried.

Each candidate solution consisted of a list of lists of (pre-evolved) ratio-templates. The list structure represented how the templates would be logically combined in a detector. This logical structure was specified at the start and only



1. Pick the next detection.
2. Find (the nearest) overlapping detection.
3. If there is an overlapping detection merge the two together and replace them with the merged version.
4. Repeat, until no overlapping detections are found.

Figure 2.7: Merging Overlapping Detections

the exact templates used could be altered. In this way we could for example try to find a good combination of two or three different templates that would yield a precise detector, which also had a good recall rate.

To evaluate how good (or bad) a detector was the following evaluation function was used:

$$score = precision * (10.0 + recall)$$

This biased the search towards precise solutions and typically means that an increase in the recall would only be accepted if the precision has not decreased significantly. The hope being to get "completely" precise solutions and then slowly increase the recall rate without adversely affecting the precision.

It may also be possible to incorporate the combining of ratio-templates into the main algorithm. It could then also be the case that we might wish to evolve simpler templates, but add an additional goal to the evolutionary process of producing combinations of templates that are also computationally efficient. However as the hillclimbing was only attempted towards the end of this project there was insufficient time to really explore such areas.

2.6.3 Heuristic Tidying

Once the detection process has occurred we are often left with several detections grouped around the same location. This is not too much of a problem when looking at these detections overlaid on an image, as we tend to automatically consider them one as detection. However if we want to use the results of the detections in a programmatic fashion (e.g. as input to another system) we need to group these detection as best we can into one detection. This was done in an iterative fashion (see Figure 2.7) and does not add too much extra cost to the detection process - as long as there are not too many detections.

Originally overlapping was considered to occur when the bounding boxes of the detections overlapped, but this does not give could results when the

detections are only overlapping at the corners. So instead the test to see if two detections overlapped was done using a circular shape, thus preventing the merging of detections that are barely overlapping. This idea could be extended further, by making the radius of the circle used smaller than the detection size and by considering how similar in size each detection is. Hopefully that would make the heuristic more robust in the face of false-detections, which might otherwise produce unexpected results after merging.

So as to evaluate the accuracy of the *actual* detection process the heuristic was not applied to any of the images shown in the results section and is merely included here as a refinement of the overall process.

Chapter 3

Experiments and Results

Now that the workings of the face detection system have been outlined it is necessary to demonstrate it in action. In order to do this we need to evolve ratio-templates that are capable of classifying face/non-face images. To do this, several different experiments were performed, with the intention of creating viable ratio-templates for use in a detector.

After the templates were evolved it also proved necessary to combine these templates, to further improve their accuracy - to the point at which a respectable face detector could be produced. Various combinations of templates, some hand-selected and some generated via hillclimbing, were tested on images containing faces. The detectors were also tested using a webcam, to help get a better feel for how they work in a "real-world" situation.

3.1 Data Sets

Two face data sets were used during experimentation. The training/evolution set consists of 450 20x20 images derived from images from the Caltech Computational Vision Archive [2] of roughly 27 different people. The test set consists of 306 20x20 images derived from the ATT Database of Faces [6] of 40 different people. Figure 3.1 a) shows some of the faces images used during evolution and Figure 3.1 b) some of those used for testing.

During evolution an initial set of roughly 600 non-face images were supplied. These initial images were taken from the backgrounds of the original Caltech images, but otherwise have no particular pattern (apart from not containing faces). The background images used during "boot-strapping" consisted of several outdoor and indoor shots, as well as some images of text (due to texts regular shape).

For evaluation, after evolution, two sets of non-face images were also used. One set was derived from the backgrounds used during training and one from a different set of background images. Both sets consisted of 10,000 20x20 images representing sub-images of various scales in the backgrounds.

(a) Example Faces Used During Evolution (partial set)

(b) Faces Used for Testing

Figure 3.1: Face Images

3.2 Experiments

In addition to choosing which candidate (out of the three) to use, each experiment used one of three fitness functions:

- "Lenient" $fitness = 0.9 * tp + 0.1 * fp$
- "Strict" $fitness = 0.1 * tp + 0.9 * fp$
- "Moderate" $fitness = 0.5 * tp + 0.5 * fp$

Where tp is the true-positive rate and fp is the false-positive rate.

In turn each experiment was also initialised using one of three region schemes:

- "Sinha" - 11 regions corresponding roughly to a face shape.
- "Rowley" - 24 regions organised in overlapping grids and rows.
- "Random" - 20 regions placed randomly over the template area.

This therefore leads us to twenty seven different experiments - one for each combination of candidate, fitness function and regions.

3.2.1 ROC Graphs

As we performed ten runs of each experiment this obviously leaves us with a lot of data to look at. To make comprehension of the data a bit easier the true-positive and false-positive rates for each experiment were plotted as ROC graphs (Receiver Operating Characteristics[8]). Two groups of data points were plotted on these graphs, one set representing the "training set" and one representing a "test set". The face images for the "training set" were those used during evolution, whereas the "test set" contained a different In both cases 10,000 non-face images were used to estimate the false-positive rates. The non-face images for the "training set" were randomly selected sub-images, of assorted scales, from the background images used during evolution. The "test set" non-face images were drawn from a different set of background images, that were not used during evolution.

3.2.2 Experimental Results

The average precision and recall on the "training" and "testing" data sets for each experiment (across 10 runs each) can be seen in Table 3.1.

Genotypes

The first thing to look at is how well the experiments using the RTCandidate and those using the MRTCandidate fare. Given that the MRTCandidate is the same as the RTCandidate, only with the addition of being able to evolve the position of its regions, it would be wise to see if this difference has any effect. The short answer being yes. In fact the difference is actually quite dramatic. For the training sets every experiment using an MRTCandidate is, on average, better in both precision and recall than its counterpart RTCandidate experiment. This trend continues for the testing set, where only one experiment

Experiment	**Average**			
	Training		**Testing**	
	Precision	**Recall**	**Precision**	**Recall**
RT Lenient Sinha	0.273	0.993	0.154	0.924
RT Lenient Rowley	0.395	0.970	0.171	0.651
RT Lenient Random	0.217	0.988	0.109	0.842
RT Strict Sinha	0.378	0.003	0.000	0.000
RT Strict Rowley	0.978	0.560	0.732	0.100
RT Strict Random	0.863	0.117	0.304	0.009
RT Moderate Sinha	0.779	0.818	0.531	0.647
RT Moderate Rowley	0.910	0.820	0.616	0.292
RT Moderate Random	0.834	0.708	0.420	0.213
MRT Lenient Sinha	0.615	**0.995**	0.379	0.846
MRT Lenient Rowley	0.531	0.988	0.301	0.766
MRT Lenient Random	0.394	0.988	0.212	0.806
MRT Strict Sinha	0.725	0.118	0.408	0.044
MRT Strict Rowley	**0.982**	0.641	**0.911**	0.177
MRT Strict Random	0.967	0.438	0.694	0.111
MRT Moderate Sinha	0.916	0.919	0.767	0.657
MRT Moderate Rowley	0.944	0.892	0.808	0.440
MRT Moderate Random	0.907	0.881	0.749	0.500
NEAT Lenient Sinha	0.283	0.989	0.162	0.929
NEAT Lenient Rowley	0.263	**0.995**	0.141	0.933
NEAT Lenient Random	0.255	0.993	0.143	**0.936**
NEAT Strict Sinha	0.970	0.572	0.804	0.189
NEAT Strict Rowley	0.962	0.606	0.810	0.251
NEAT Strict Random	0.950	0.527	0.799	0.207
NEAT Moderate Sinha	0.775	0.879	0.538	0.624
NEAT Moderate Rowley	0.849	0.883	0.613	0.499
NEAT Moderate Random	0.806	0.898	0.570	0.607

$Precision = TP/(TP + FP)$ $Recall = TP/P$

Table 3.1: Average Precision and Recall

yields better average recall for the RTCandidate and even then the precision for that experiment is worse.

The NEATCandidate experiments appear to have performed more or less better than the RTCandidate experiments, but do not dominate them in the same way as the MRTCandidate experiment. This is slightly disappointing as the slightly more dynamic nature of the genotypes encoding scheme is appealing. However one suspects that this may also be part of the reason for the relative failure - more work is probably needed to carefully tune this genotype.

We will concentrate on the nine experiments that make use of the MRTCandidate, as these have the best overall results.

Fitness Function and Regions

As might be expected the experiments using the "strict" fitness function, biased against false-positives, have for the most part the highest precision, but also the lowest precision. Conversely the exact opposite is true for those using the "lenient" fitness function. This leaves the "moderate" experiments in the middle, which might not seem very useful. Seeing as we have not evolved any detectors that have both the best precision and best recall it would seem we have no clear winner. However all is not lost.

Looking at the ROC graphs for the MRTCandidate experiments (Figures 3.2-3.10) can help us decide which conditions yielded the best templates and hence which templates to consider for use as (part of) a detector. Each of the graphs contains two data sets. One represents the true-positive and false-positive rates on the training set (squares) and one on the test set (circles). Roughly speaking a "better" detector will have as many of these points plotted nearer to the top-left hand corner of the graph - the point at which we have a perfect detector[1]. The concept of distance/nearness can be applied differently depending on what kind of behaviour we are hoping for. In this case a low false-positive rate is desirable, this is partly due to personal preference, but as mentioned previously a false-positive rate in the region of 1 in 1,000,000 is needed "for a face detector to be practical for real applications"[26]. Taking into account this preference we can make some more informed decisions.

The ROC graphs for the lenient experiments (Figures 3.2-3.4) have most of their points plotted closer to the top of the graph. In fact no single point has worse than a 0.6 true-positive rate. The best graph (for the Sinha regions - Figure 3.2) would seem to be quite well squeezed into the top-left corner, but even the best point still has a "high" [2] false-positive rate.

The strict experiments (Figures 3.5-3.7) are actually worse than the lenient ones, despite having very low false-positive rates. They are let down by their abysmal true-positive rates. In particular the points for the test set are usually very low down, never rising much above 0.4. This makes the combining of these templates, discussed later, more difficult - attempting to further reduce the false-positive rate is likely to lead to an even worse true-positive rate.

This leads us to the moderate experiments (Figures 3.8-3.10). The ROC graphs for these look almost like those for the strict experiments, but with one important difference: the true-positive rate is much better. As in the lenient experiments the best of these seems to be the Sinha variant (Figure 3.8). If we

[1] i.e. a false-positive rate of 0.0 and a true-positive rate of 1.0

[2] About 0.01-0.02

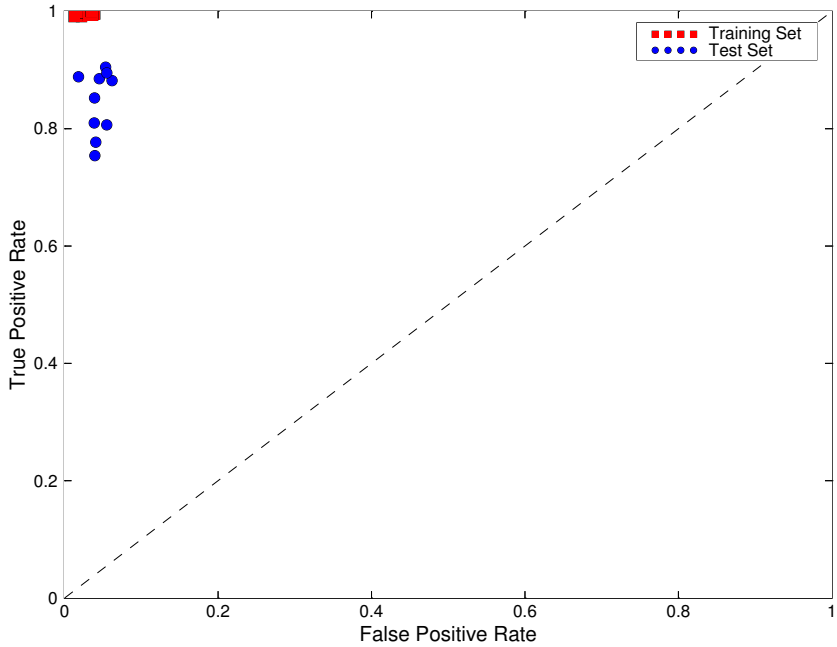


Figure 3.2: ROC Graph - MRT Lenient Sinha

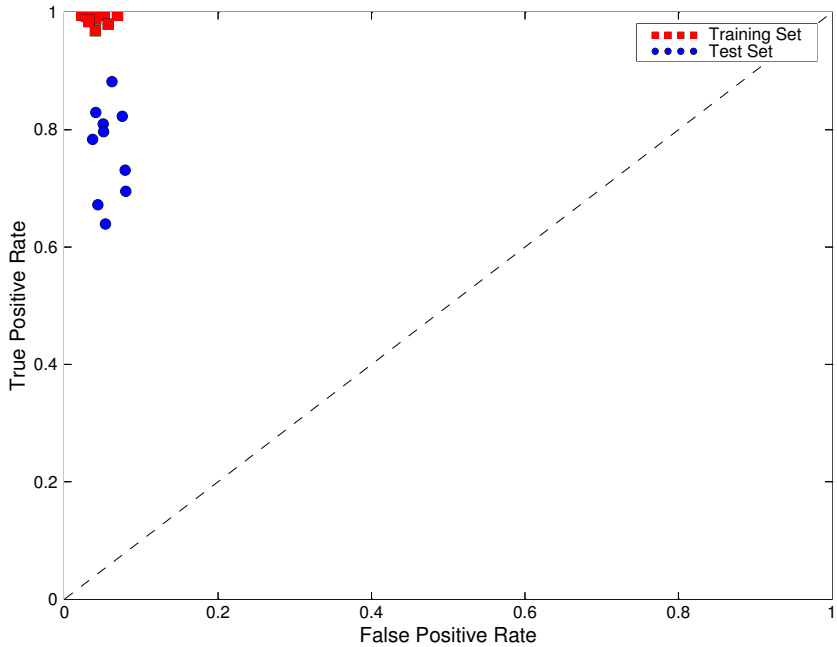


Figure 3.3: ROC Graph - MRT Lenient Rowley

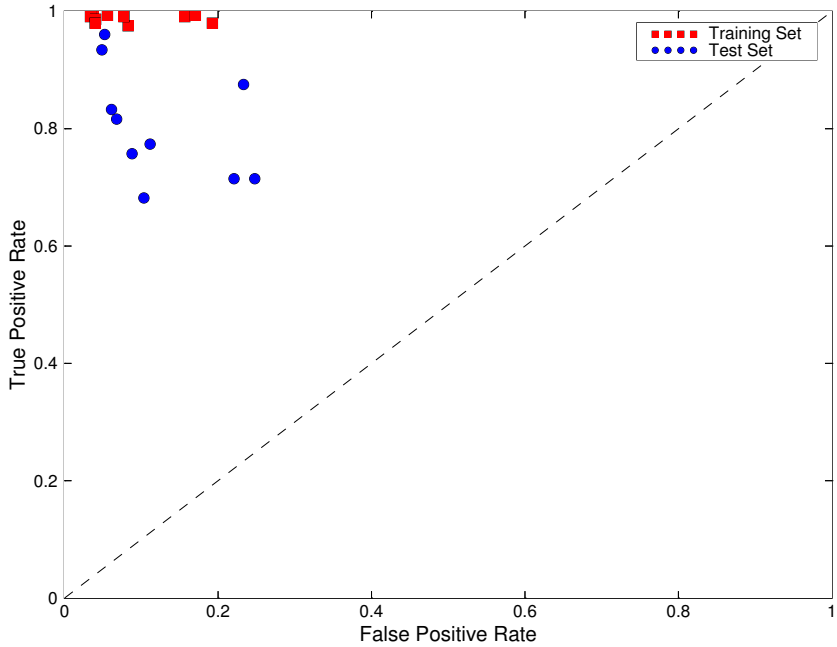


Figure 3.4: ROC Graph - MRT Lenient Random

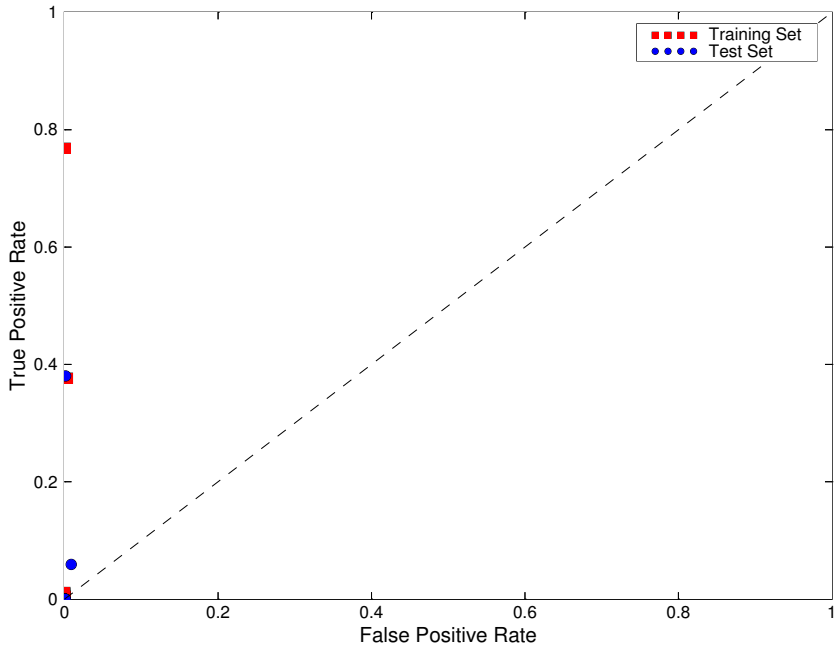


Figure 3.5: ROC Graph - MRT Strict Sinha

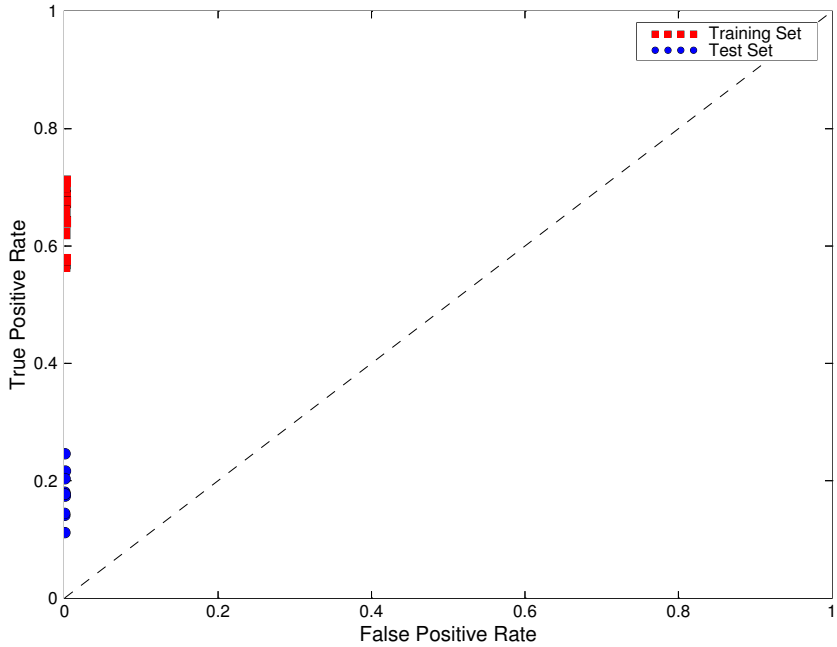


Figure 3.6: ROC Graph - MRT Strict Rowley

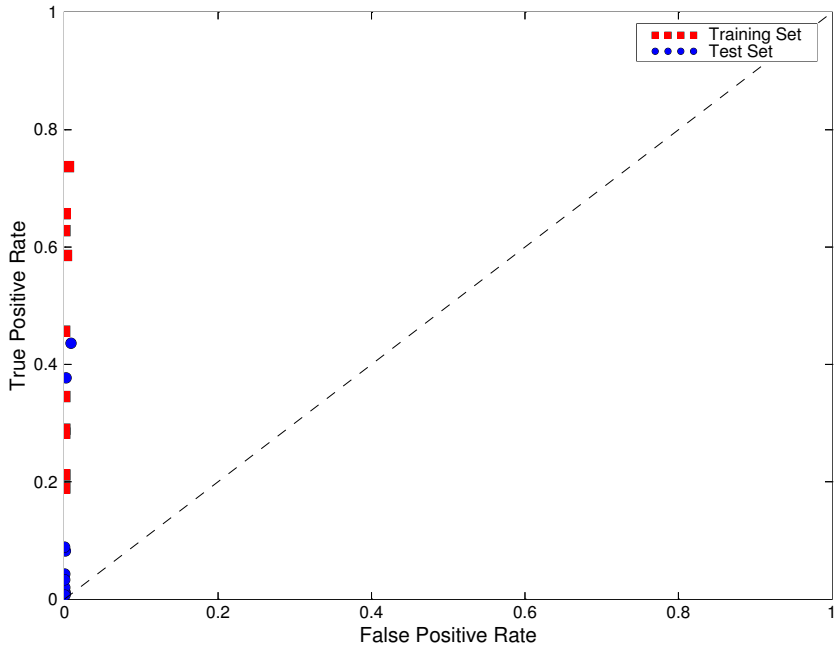


Figure 3.7: ROC Graph - MRT Strict Random

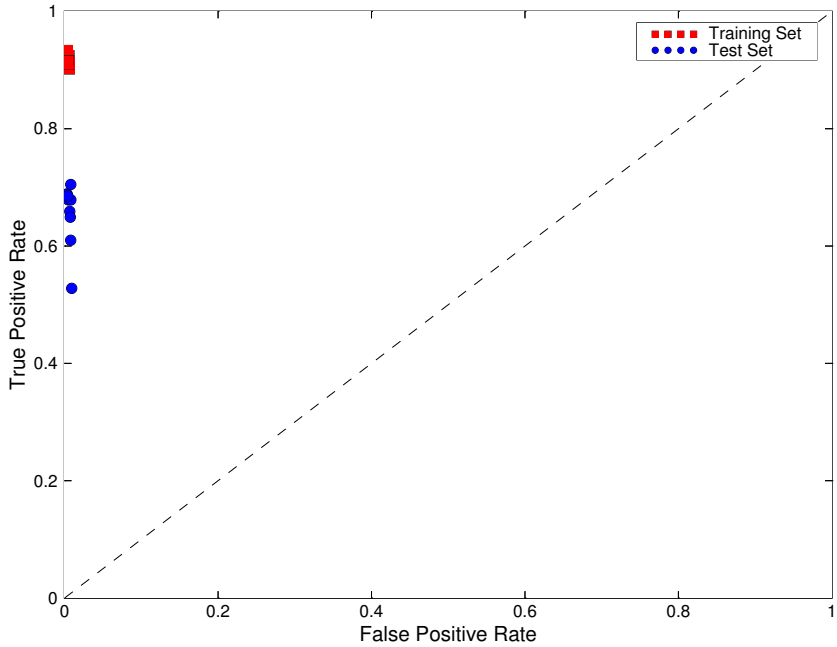


Figure 3.8: ROC Graph - MRT Moderate Sinha

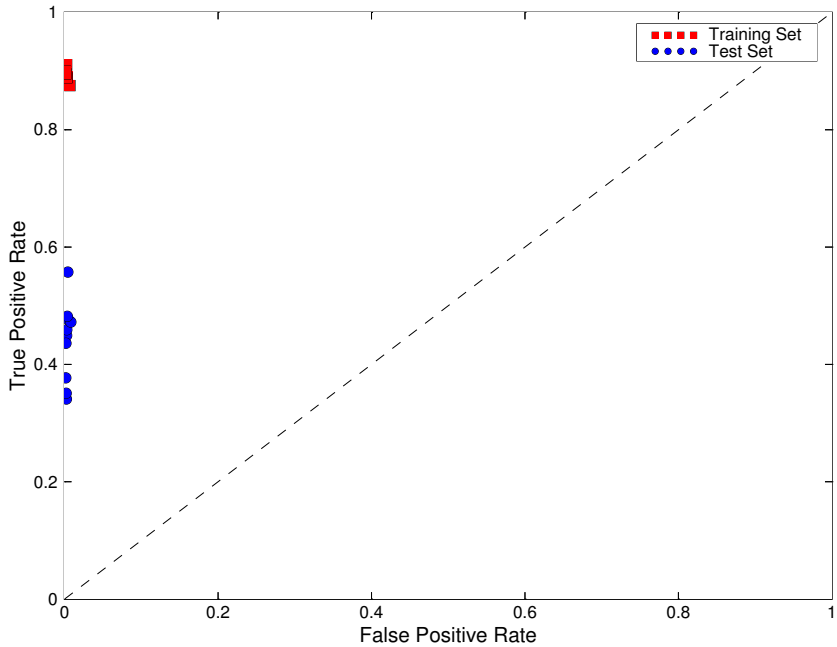


Figure 3.9: ROC Graph - MRT Moderate Rowley

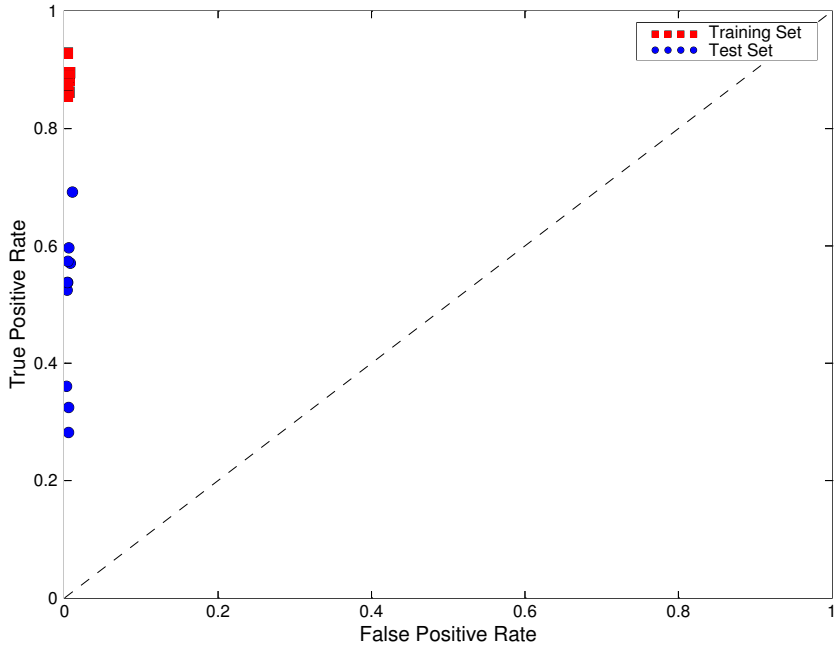


Figure 3.10: ROC Graph - MRT Moderate Random

compare these two graphs (Figure 3.2 vs. Figure 3.8) we see that whilst the true-positive rates for the lenient experiment are better they are not orders of magnitude better then those in the moderate experiment. The same cannot be said for the false-positive rates. At the scale shown the moderate experiments seem to have an almost zero false-positive rate and the lenient experiments false-positive rates are quite obvious.

In short the moderate experiments have both good false-positive *and* true-positive rates which collectively leads to a "better" detector. These templates will form a good basis for the work discussed in the following section.

3.3 Improving the Results

At this point we have several templates - some of which work reasonably well. However they only work reasonably well on the task of discriminating between face and non-face images. For our real task of detecting positions (and scales) of faces in images they will need to be further improved.

3.3.1 AND'ing

Table 3.2 shows the precision and recall rates when all ten templates evolved in an experiment are AND'ed together to yield the final detection result. By comparison with Table 3.1 (averages for each experiment) one can see that in general the precision has increased, but the recall has reduced also. The precision (how often a detection is actually a true-positive) improves, because each evolved template will tend to result in *different* false-positives, thus reducing

Experiment	AND'ed			
	Training		Testing	
	Precision	Recall	Precision	Recall
RT Lenient Sinha	0.766	0.953	0.569	**0.807**
RT Lenient Rowley	0.898	0.922	0.616	0.374
RT Lenient Random	0.986	0.920	0.950	0.370
RT Strict Sinha	**1.000**	0.000	**1.000**	0.000
RT Strict Rowley	**1.000**	0.307	**1.000**	0.003
RT Strict Random	**1.000**	0.000	**1.000**	0.000
RT Moderate Sinha	0.993	0.610	0.901	0.479
RT Moderate Rowley	0.994	0.704	0.875	0.092
RT Moderate Random	**1.000**	0.332	**1.000**	0.007
MRT Lenient Sinha	0.998	0.967	0.984	0.610
MRT Lenient Rowley	0.998	0.942	0.991	0.380
MRT Lenient Random	**1.000**	0.942	**1.000**	0.430
MRT Strict Sinha	**1.000**	0.000	**1.000**	0.000
MRT Strict Rowley	**1.000**	0.314	**1.000**	0.010
MRT Strict Random	**1.000**	0.056	**1.000**	0.000
MRT Moderate Sinha	**1.000**	0.724	**1.000**	0.305
MRT Moderate Rowley	**1.000**	0.762	**1.000**	0.108
MRT Moderate Random	**1.000**	0.679	**1.000**	0.085
NEAT Lenient Sinha	0.991	0.949	0.965	0.728
NEAT Lenient Rowley	0.962	**0.971**	0.924	0.718
NEAT Lenient Random	0.935	0.960	0.856	0.780
NEAT Strict Sinha	**1.000**	0.167	**1.000**	0.003
NEAT Strict Rowley	**1.000**	0.205	**1.000**	0.026
NEAT Strict Random	**1.000**	0.085	**1.000**	0.003
NEAT Moderate Sinha	**1.000**	0.684	**1.000**	0.174
NEAT Moderate Rowley	**1.000**	0.733	0.978	0.144
NEAT Moderate Random	**1.000**	0.757	**1.000**	0.213

$Precision = TP/(TP + FP)$ $Recall = TP/P$

Table 3.2: AND'ed Precision and Recall

the collective false-positive rate and thereby improving the precision (see Figure 2.5). Having a high precision detector is important, as then we can be pretty certain that anything it *does* detect is actually a face. However the recall (how many faces are actually detected) reduces also, which means we are also less likely to have any detections.

In this case, the use of ten templates AND'ed together often gives us very good precision levels, but we can probably use slightly fewer templates and still achieve similar precision levels and yet actually detect more faces successfully.

3.3.2 OR'ing

Table 3.3, for comparison, shows the effects of OR'ing the outputs of all ten templates in an experiment together. This gives us the opposite effect of the AND'ing shown in Table 3.2 - namely that the precision is reduced (often mas-

Experiment	OR'ed			
	Training		Testing	
	Precision	Recall	Precision	Recall
RT Lenient Sinha	0.104	**1.000**	0.062	0.993
RT Lenient Rowley	0.157	0.991	0.072	0.862
RT Lenient Random	0.043	**1.000**	0.030	**1.000**
RT Strict Sinha	0.118	0.009	0.000	0.000
RT Strict Rowley	0.887	0.786	0.550	0.289
RT Strict Random	0.828	0.537	0.342	0.085
RT Moderate Sinha	0.423	0.953	0.218	0.833
RT Moderate Rowley	0.632	0.900	0.325	0.567
RT Moderate Random	0.479	0.940	0.231	0.679
MRT Lenient Sinha	0.240	**1.000**	0.121	0.961
MRT Lenient Rowley	0.172	**1.000**	0.091	0.984
MRT Lenient Random	0.096	**1.000**	0.052	0.997
MRT Strict Sinha	0.860	0.822	0.508	0.403
MRT Strict Rowley	**0.893**	0.857	**0.774**	0.495
MRT Strict Random	0.823	0.878	0.592	0.610
MRT Moderate Sinha	0.595	0.987	0.373	0.918
MRT Moderate Rowley	0.695	0.960	0.469	0.751
MRT Moderate Random	0.575	0.964	0.386	0.843
NEAT Lenient Sinha	0.083	**1.000**	0.046	**1.000**
NEAT Lenient Rowley	0.093	**1.000**	0.049	0.997
NEAT Lenient Random	0.086	**1.000**	0.048	**1.000**
NEAT Strict Sinha	0.863	0.855	0.586	0.570
NEAT Strict Rowley	0.814	0.884	0.550	0.669
NEAT Strict Random	0.787	0.862	0.548	0.620
NEAT Moderate Sinha	0.348	0.976	0.196	0.954
NEAT Moderate Rowley	0.499	0.951	0.282	0.813
NEAT Moderate Random	0.390	0.964	0.215	0.921

$Precision = TP/(TP + FP)$ $Recall = TP/P$

Table 3.3: OR'd Precision and Recall

sively), but the recall increases. Simply OR'ing the templates in this way is also not enough, so we need to try another approach.

3.3.3 Logical Grouping

Next combining templates using both AND'ing (to increase precision) and OR'ing (to increase recall) was tried. Templates were formed into groups that were AND'ed together, then the results of each group were OR'ed to give the final result. Each group should then individually have a very low false-positive rate. By OR'ing the groups together we can increase the collective recall rate, whilst hopefully still maintaining a high precision.

Initially the selection of these groups was done crudely by hand. For example the results of combining the three MRT moderate experiments (Figures 3.8-3.10) by AND'ing each experiment, as in Table 3.2, and then OR'ing each AND'ed

experiment together (as in Figure 2.6) can be seen in Figures 3.11-3.14. The figures show the combined detector being tested on the source images [3] for the training set. As can be seen this looks reasonably convincing. There are roughly two false-positives (in Figures 3.11 and 3.14) and seventy eight false-negatives. Out of 450 images this still represents a recall rate of:

$$recall = (450 - 78)/450 = 0.827 \text{ (3 d.p.)}$$

and a precision rate of:

$$precision = (450 - 78)/(450 - 78 + 2) = 0.995 \text{ (3 d.p.)}$$

Which is at the very least what one would hope for when testing on images used for the training set. It also also worth noting that a lot of the false-negatives occur on faces that are either lit from behind or else tilted slightly (which was part of the reason for pre-processing the faces for training by aligning them).

Next the combined detector was tested on a series of images containing multiple faces [4]. The results for some of these detections can be seen in Figure 3.15 a). On most of the multiple face images used the false-positive rate was very low, however the true-positive rate was also quite low. So the detectors precision was high, but its recall quite low. The detector seemed accurate, but overly conservative.

3.3.4 Detecting Faces in (not quite) Real-time

One of the original intentions of this project was to be able to successfully locate and/or track faces in real-time with a camera of some sort. So to this end it seemed remiss not to try out the various evolved detectors on a web-cam. Initially the detector used previously in Section 3.3.3 was tested, but as noted it proved too conservative and produced no detections! So in the end the number of templates was paired down slightly, resulting in a detector comprising of three groups of three templates. This allowed the detector to successfully detect the author's face in "interactive" time [5]. This detection was recorded for posterity and is available as a short movie clip [6], a few frames of which can be seen in Figure 3.16 a).

3.3.5 Hillclimbing

The precision and recall rates for hillclimbing were calculated using both the training and testing faces from the evolutionary runs and 40,000 random background images.

Up to a certain point the hillclimbing approach seemed to work. The precision and recall rates being reported were *much* better than those of either the individual templates and the combinations used previously. However it seemed that even an apparently "perfect" precision on the data used, did not always entail a high precision on real data. The difference is apparent in Figure 3.15

[3] The images that the training set faces (20x20 pixel images of just the faces) were extracted from

[4] Mainly images from Rowley et al's test set.

[5] Not "real-time", but a few seconds for each detection. i.e. fast enough for someone to interact with.

[6] http://studentweb.cs.bham.ac.uk/∼msc37jxm/project/webcam.mpg

Figure 3.11: Training Data Detections, using simple combination of templates

Figure 3.12: Training Data Detections, using simple combination of templates (continued)

Figure 3.13: Training Data Detections, using simple combination of templates (continued)

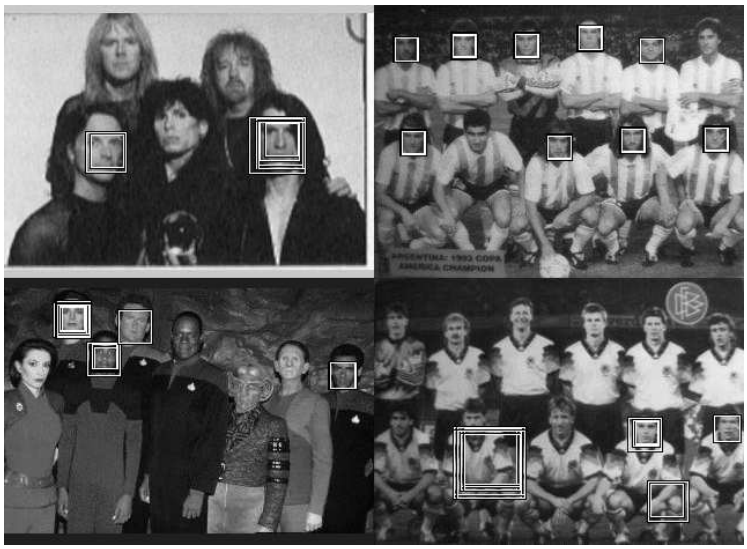
Figure 3.14: Training Data Detections, using simple combination of templates (continued)

b), when compared with Figure 3.15 a). One can clearly see that the precision is much worse. However the recall rate does generally seem higher, as more faces seem to be correctly picked out. As Rowley et al [20] note it is difficult to find a representative set of "non-face" images for training. So perhaps it would be necessary to perform hillclimbing using the "boot-strapping" technique used during evolution. Until this is done it will not become clear as to whether we require a "better" algorithm to select the templates.

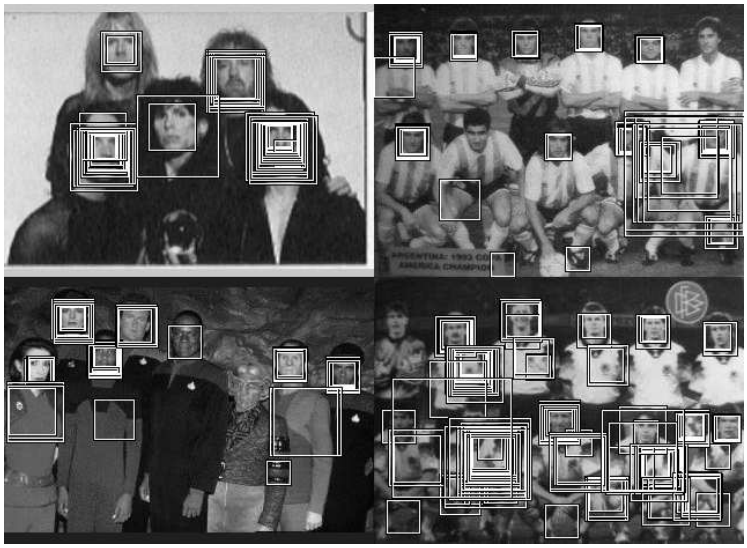
Real-time Detection using a Hillclimbed Detector

To further compare how the current use of hillclimbing to select templates compares with manual selection, the detector previously used was tested with a webcam. As this was performed under slightly different lighting conditions, the hand-selected detector was tested again and was found to behave as before.

Some sample frames can be seen in Figure 3.16 b). When they are compared against the hand-selected detector (Figure 3.16 a)) we see that it is (again) not quite as good. In particular it persistently incorrectly detects faces in at least two positions and "loses" the author's face when it appears it should not. However this is an improvement over just using a single template. So it does show that automation of this "last step" could be appropriate - with a little more work.



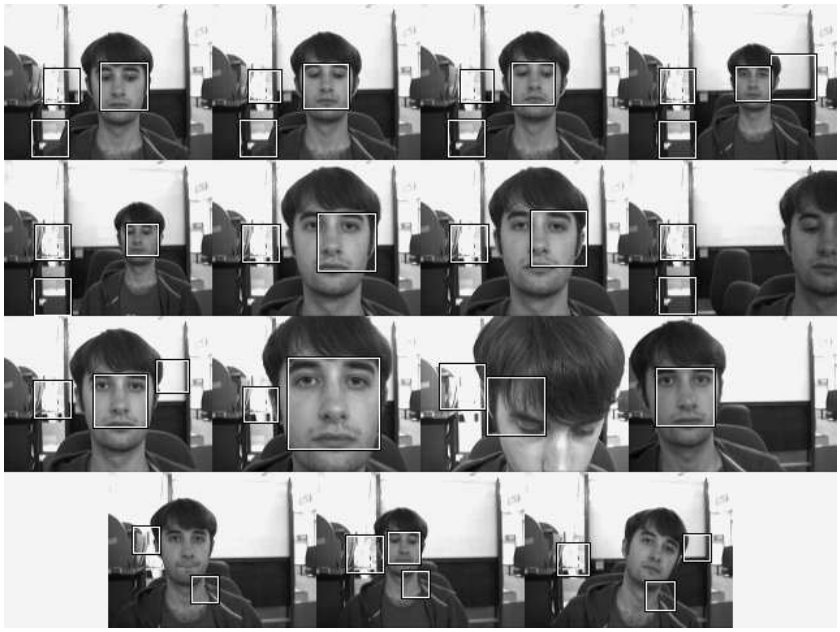
(a) Hand-Selected Detector



(b) Hillclimbed Detector

Figure 3.15: Testing on Group Photos

(a) Hand-Selected Detector



(b) Hillclimbed Detector

Figure 3.16: The Author's Face being Detected

Chapter 4

Extensions, Improvements and Conclusion

4.1 "Better" Genotypes

The MRTCandidate genotype proved to be the most effective genotype, in terms of producing good classifiers. It seemed that allowing the evolution of both the ratios *and* regions of a ratio-template, gave the MRTCandidate a real competitive edge over the RTCandidate, which could only evolve ratios and not regions. However the MRTCandidate genotype is quite basic and not very dynamic - only allowing the evolution of a fixed number of regions and ratios. The NEATCandidate went some way to address the lack of dynamism, but did not perform as well. Obviously some work needs to be done to address this.

4.1.1 Enforce Symmetry

For the task of detecting upright faces there would be one obvious feature to exploit, that of symmetry[13]. If a genotype only encoded half of the ratio-template (with the other half being mirrored), performance could be improved by constraining the search space. The ratio-templates produced would *always* be symmetric, which would most likely be a huge benefit for detecting symmetric objects such as faces. Conversely un-symmetric objects would less likely be classified incorrectly as faces, lowering the false-positive rate.

As only the genotype would be enforcing the symmetry this means that the system as a whole would remain quite general. By using different genotypes, one could still evolve detectors for different purposes. e.g. detecting faces in profile.

4.1.2 Enforce Neighbourliness

The original idea behind ratio-templates was to examine local differences in brightness[24]. The current genotypes only loosely encourage this, with mutation biased towards selecting regions near to each other, for use in ratios. As with enforcing symmetry it may help reduce the search space. The total number of ratios to be examined would be limited and the examination of many spurious

ratios would be eliminated, as regions that are far apart are less likely to share an invariant relationship..

With a genotype that does not alter the positions of regions this would be straightforward, as the information could be encoded ahead of time. Even if the initial positions can change, the information would still be valid, as long as the position changes are not too radical. Otherwise it would be a case of being able to decide when two regions are neighbours, via programmatic means, and deciding how to deal with situations where some regions have no neighbours.

4.2 Better, Faster, Stronger Classifiers

The current evolved ratio-templates evolved are able to discriminate, to a reasonable, between face and non-face images. However to get a decent level of classification performance they must be (manually) combined. This potentially slows down the whole system and requires a level of human interaction that could probably be automated.

4.2.1 Cascading Ratios

Viola and Jones[26] "cascade" their feature detectors together, so that as soon as one fails to match no others need examine, meaning non-face images can quickly be classified as such. Extending this idea to a ratio-template would be quite easy and has in fact been tried by Scassellati[23], wherein he identified eleven "essential" ratios, of which ten must be matched. As soon as two of the essential ratios fail to match, the image can be rejected.

One could take this step even further and make the order of the ratios matter. As each ratio is examined its weighting would be added or subtracted[1] from a running total. If at any point the running total falls below zero then the entire match would fail. The absence of a single ratio, with a large weighting (evaluated early), might then be enough to halt all further (potentially wasted) ratio evaluations.

By including, as part of the genetic algorithms fitness function, some measure of how many ratios need evaluating before a non-face image is classified as such, it may be possible to produce ratio-templates that are very efficient. The ordering of ratios could either be controlled by their weightings (larger weights implying earlier evaluations) or else independently, but either way could result in large performance gains.

4.2.2 Combining Ratio-Templates

Although some work was attempted to automate the combination of multiple ratio-templates (see Section 3.3.5) this proved unsatisfactory. At the very least the "boot-strapping" technique, used during evolution, also needs to be applied to the hillclimbing process, to help get over the problem of providing a representative set of non-face images[20].

Alternatively it might be possible to integrate the combining stage with the evolutionary process. This could possibly be achieved by cooperatively co-evolving the ratio-templates with genotypes representing combinations of those

[1] depending on whether the ratio is present or absent in the image.

same templates. To do this we would need to ensure a certain amount of variation in the behaviour of the templates being evolved, otherwise combining them will have little benefit. We would either need to have multiple populations of separately evolving templates or else use genetic algorithms taylored to maintaining diversity[2].

4.3 Handle Rotation

Rowley, Baluja and Kanade[21] extended their "upright frontal face detection system"[20] to deal with faces rotated in the image plane. This was achieved using a second neural network as a "router network"[20]. The job of the router network was merely to indicate the current angle of rotation of a face presented to it. This information would then be used to align the image so that the face was upright. Then the newly rotated image would be presented to the classifier network to decide if it was actually a face image or not. This extension was partly motivated by the fact that "people expect face detection systems to be able to detect rotated faces"[20].

The way Rowley et al. added the ability to detect rotated faces to their system is very interesting. The extra network is used in quite a modular fashion. In principle the current system could be augmented by the addition of such a router network, but it would be interesting to attempt a similar feat without a neural network. Might it be possible to create a ratio-template that could perform similar duties?

4.4 Conclusion

I have implemented, in C++ and Python, a face-detection system based on ratio-templates. Using genetic algorithms I have evolved ratio-templates as face/non-face classifiers. These evolved ratio-templates, whilst not individually strong enough as classifiers for the face-detection task, when combined produced acceptably accurate and precise classifiers. Testing of these combined classifiers, on various images and using a web-cam, shows that they have a sufficiently low false-positive rate and respectible true-positive rates.

As a proof-of-concept and for the development time-frame involved, this project has been successfull. The system definitely *works* and with further improvement could potentially form the basis of a very robust, efficient face detection system.

[2] e.g. niching/fitness sharing etc

References

[1] K. Anderson and P.W. McOwan. Robust real-time face tracker for cluttered environments. *Computer Vision and Image Understanding*, 2004.

[2] Caltech Computational Vision Archive. Human face (front) dataset. http://www.vision.caltech.edu/html-files/archive.html.

[3] V. Atienza and J.M. Valiente. Face tracking algorithm using grey-level images. In *IASTED Int. Conf. On Signal Processing and Communications (SPC'2000)*, pages 507–512, 2004.

[4] D. Beazley. SWIG: Simplified wrapper and interface generator. http://www.swig.org/.

[5] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University, 1995.

[6] AT&T Laboratories Cambridge. Database of faces. http://www.uk.research.att.com/facedatabase.html.

[7] B.A. Draper, K. Back, M.S. Bartlett, and J.R. Beveridge. Recognizing faces with pca and ica. *Computer Vision and Image Understanding*, 91:115–137, 2003.

[8] T. Fawcett. ROC graphs: Notes and practical considerations for researchers. Technical Report HPL-2003-4, HP Laboratories, 2004.

[9] J.H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 5th edition, 1991.

[10] D. Howard, S.C. Roberts, and R. Brankin. Target detection in sar imagery by genetic programming. *Advances in Engineering Software*, 30:303–311, 1999.

[11] L. Huang, A. Shimizu, Y. Hagihara, and H. Kobatake. Face detection from cluttered images using a polynomial neural network. *Neurocomputing*, 51:197–211, 2003.

[12] Intel. Open computer vision library. http://www.intel.com/research/mrl/research/opencv/.

[13] K.J. Kirchberg, O. Jesorsky, and R.W.Frischolz. Genetic model optimization for hausdorff distance-based face localization, lncs 2359. In *Internation ECCV 2002 Workshop on Biometric Authentication*, pages 103–111. Springer-Verlag, 2002.

[14] A.Z. Kouzani. Locating human faces within images. *Computer Vision and Image Understanding*, 91:247–279, 2003.

[15] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[16] M. Lutz. *Programming Python.* O'Reilly, 2nd edition, 2001.

[17] M. Mirmehdi, P.L. Palmer, and J. Kittler. Robust line segment extraction using genetic algorithms. In *Sixth IEE International Conference on Image Processing and its Applications*, pages 141–145. IEE Publications, July 1997.

[18] M. Mirmehdi, P.L. Palmer, and J. Kittler. Optimising the complete image feature extraction chain. In *Third Asian Conference on Computer Vision*, volume 2, pages 307–314. Springer Verlag, 1998.

[19] S. Perkins and Gillian Hayes. Evolving complex visual behaviours using genetic programming and shaping. *Interdisciplinary Approaches to Robot Learning*, June 2000.

[20] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:23–38, 1998.

[21] H. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network-based face detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 1998.

[22] O. Sachs. *The Man who Mistook his Wife for a Hat.* Picador, 1986.

[23] B. Scassellati. Eye finding via face detection for a foveated, active vision system. In *AAAI 98*, 1998.

[24] P. Sinha. *Perceiving and Recognizing three-dimensional forms.* PhD thesis, Massachusetts Institue of Technology, 1996.

[25] K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[26] P. Viola and M.J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

[27] S. Wiegand, C. Igel, and U.Handmann. Evolutionary optimization of neural networks for face detection. In *12th European Symposium on Artificial Neural Networks (ESANN 2004).* d-side publications, 2004.

[28] M. Zhang and W. Smart. Multiclass object classification using genetic programming. *EvoWorkshops 2004, LNCS 3005*, pages 369–378, 2004.

Appendix A

Electronic Resources

Electronic resources related to this project such as the source-code, poster and project report will be made available at:

`http://studentweb.cs.bham.ac.uk/~msc37jxm/project/`