Face Detection using Neural Networks

Phil Brimblecombe
URN: 1046063

Project Summary

Face detection is a fundamental part of many face recognition systems, due to its ability to focus computational resources on the part of an image containing a face. The task of detecting and locating human faces in arbitrary images is complex due to the variability present across human faces, including skin colour, pose, expression, position and orientation, and the presence of 'facial furniture' such as glasses or facial hair. Differences in camera gain, lighting conditions and image resolution further complicate the situation. A survey of some of the different approaches to the problem has been included in this report as an introduction to the area. An implementation of one of the identified image-based approaches to the problem using neural networks, has been included, explained and analysed. Various parameters choices are investigated to confirm an optimal set of operational parameters. Minor adjustments were made to the detection system, reflecting some of the identified limitations, leading to small but significant improvements. The detection rate was increased from 29% to 44% on the first test set, and from 44% to 57% on the second test set, at the expense of an increase in the number of false positives. A summary of the findings is included and details of further work and extensions to the project are suggested, along with a comprehensive appendix.

Contents

1. Introduction

Pattern recognition is a modern day machine intelligence problem with numerous applications in a wide field, including Face recognition, Character recognition, Speech recognition as well as other types of object recognition. The field of pattern recognition is still very much in it is infancy, although in recent years some of the barriers that hampered such automated pattern recognition systems have been lifted due to advances in computer hardware providing machines capable of faster and more complex computation.

Face recognition, although a trivial task for the human brain has proved to be extremely difficult to imitate artificially. It is commonly used in applications such as human-machine interfaces and automatic access control systems. Face recognition involves comparing an image with a database of stored faces in order to identify the individual in that input image. The related task of face detection has direct relevance to face recognition because images must be analysed and faces identified, before they can be recognised. Detecting faces in an image can also help to focus the computational resources of the face recognition system, optimising the systems speed and performance.

Face detection involves separating image windows into two classes; one containing faces (targets), and one containing the background (clutter). It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin colour and facial expression. The problem is further complicated by differing lighting conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise. An ideal face detector would therefore be able to detect the presence of any face under any set of lighting conditions, upon any background. For basic pattern recognition systems, some of these effects can be avoided by assuming and ensuring a uniform background and fixed uniform lighting conditions. This assumption is acceptable for some applications such as the automated separation of nuts from screws on a production line, where lighting conditions can be controlled, and the image background will be uniform. For many applications however, this is unsuitable, and systems must be designed to accurately classify images subject to a variety unpredictable conditions.

A variety of different face detection techniques exist, but all can be represented by the same basic model, depicted in figure 1.

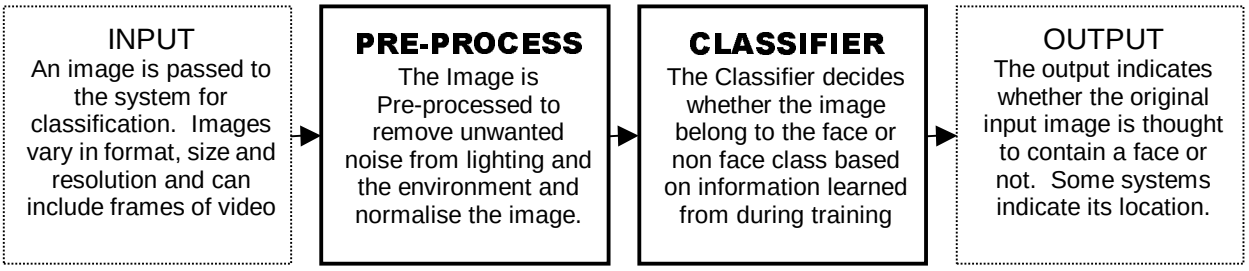


Figure 1 – A generic representation of a face detection system

Each technique takes a slightly different approach to the face detection problem, and although most produce encouraging results, they are not without their limitations. The next section outlines the various approaches, and gives six examples of different face detection systems developed.

2. Literature Survey

2.1 Face Detection Approaches – An Overview

Hjelmås and Low [1] conducted a survey on face detection techniques, and identified two broad categories that separate the various approaches, appropriately named feature-based and image-based approaches. Each category will be explained, and the work completed will be presented, providing a brief yet thorough overview of the various face detection techniques.

2.1.1 Feature-Based Approach

Hjelmås and Low [1] divide the group of feature-based systems into three sub-categories: low-level analysis, feature analysis, and active shape models.

Low-level Analysis

Low-level Analysis deals with the segmentation of visual features using the various properties of the pixels, predominantly gray-scale or colour.

Edge representation (detecting changes in pixel properties) was first implemented by Sakai et al [2] for detecting facial features in line drawings. Craw et al [3] developed this further to trace a human head outline, allowing feature analysis to be constrained to within the head outline. Various operators are used to detect the presence of an edge, including the Sobel operator, the Marr-Hildreth operator, and a variety of first and second derivatives of Gaussians. All edge-based techniques rely on labelled edges which are matched to a face model for verification. Govindaraju [4] labelled edges as left side, hairline, and right side, developing a system capable of detecting 76% of faces in a set of 60 images with complex backgrounds, with an average of two false alarms per image.

Gray Information can be used to identify various facial features. Generally Eyebrows, pupils and lips appear darker than surrounding regions, and thus extraction algorithms can search for local minima. In contrast, local maxima can be used to indicate the bright facial spots such as nose tips [5]. Detection is then performed using low-level GRAY-scale thresholding. Yang and Huang [6] explore

grayscale behaviour using an image pyramid (similar to that used later in the project), where an image is resized. Using the hypothesis that at low resolutions face regions will become uniform, face candidates are established which are verified by the existence of local minima at high resolution, representing facial features.

Colour contains extra dimensions which can help differentiate two regions which may contain similar gray information but appear very different in colour space. It was found that different skin colour gives rise to a tight cluster in colour space, thus colour composition of human skin differs little across individuals, regardless of race. The most widely used colour model is RGB, although there are many others that exist and have been used (a comparison can be found in Terrillon et al [7]).

Motion Information (where available) can be used to assist in the detection of human faces, using the principle that, if using a fixed camera, the “background clutter” will remain somewhat static, relative any “moving object”. A straightforward way to achieve motion segmentation is by frame difference analysis. Thresholding accumulated frame differences is used to detect faces or facial features. Another way to measure motion is through the estimation of moving image contours, a technique that has proven to be more reliable, particularly when the motion is insignificant.

Feature Analysis

Low-level analysis introduces ambiguity which can be solved by high level feature analysis, often through the use of some additional knowledge about the face. There are two approaches for the application of this additional knowledge (commonly face geometry).

The first involves sequential feature searching strategies based on the relative positioning of individual facial features. Initially prominent facial features are determined which allows less prominent features to be hypothesised (for example a pair of dark regions found in the face area increases the confidence of facial existence). The facial feature extraction algorithm [8], is a good example of feature searching, achieving 82% accuracy with invariance to gray and colour information, failing to detect faces with glasses and hair covering the forehead. A similar system proposed by Jeng et al [9] reported an 86% detection rate.

The second technique, constellation analysis, is less rigid and is more capable of locating faces of various poses in complex backgrounds. It groups facial features in face-like constellations, using robust modelling methods such as statistical analysis. Burl et al [10] use statistical shape theory on features detected from a multi-scale Gaussian derivative filter, capable of detecting 84% of faces, with some invariance to missing features, translation, rotation and scale. Probabilistic face models based on multiple face appearance have also been used in many systems including Yow and Cipolla's model [11] reporting a 92% detection rate.

Active Shape Model

Active shape models represent the actual physical and hence higher-level appearance of features. These models are released near to a feature, such that they interact with the local image, deforming to take the shape of the feature. There are three types of active shape models that have been used throughout the literature: snakes, deformable templates and smart snakes.

Snakes, or Active Contours, are commonly used to create a head boundary. Created nearby, they lock on to nearby edges, eventually assuming the shape of the head. The evolution of a snake is achieved by minimising an energy function, which consists of the sum of an internal energy function, defining its natural evolution (typically shrinking or expanding), and an external energy function, which counteracts the internal energy enabling the contours to deviate from the natural evolution. Energy minimisation can be obtained by optimisation techniques such as the steepest gradient descent although the additional computational demands have encouraged others to use faster iteration methods [12], and Lam and Yan [13]. Snakes are prone to becoming trapped on false image features and are not efficient in extracting non-convex features due to their tendency to attain minimum curvature. Gunn and Nixon [14], introduced a parameterised snake model to overcome these limitations by using dual integrated snakes.

Deformable Templates can be used as an extension to the snake models. Yuille et al [15] incorporated global information of the eye to improve the extraction process, using a deformable eye template. Once established near an eye feature, would deform itself, toward optimal feature boundaries using steepest gradient descent minimisation as the deformation mechanism. One limitation of such techniques is that they are sensitive to initial placement. Yuille et al [15] showed promising results when the template was placed below the eye, but noted that if place above the eye,

the template may be attracted towards the eyebrow instead. A further limitation is the processing time attempts have been made to improve this. Besides eye templates, the use of mouth templates has also been introduced.

Smart Snakes, or Point Distributed Models (PDMs) are compact parameterised descriptions of a shape based upon statistics. They use Principle Components Analysis (PCA) to construct a linear flexible model from variations of the features in a training set. Face PDM was first developed by Lantis et al [16] as a flexible model with promising results (95% detection rate). Multiple faces can be detected, tests have shown that partial occlusion is not a problem as other features are still available to contribute to a global optimal solution.

2.1.2 Image-Based Approach

Face detection by explicit modelling of facial features is a very rigid approach which has been shown to be troubled by the unpredictability of faces and environmental conditions. There is a need for more robust techniques, capable of performing in hostile environments, such as detecting multiple faces with clutter-intensive backgrounds. This has inspired a new research area in which face detection is treated as a general pattern recognition problem. Whereas face recognition deals with recognising the face, face detectors must recognise an object as a face, from examples. This eliminates the problem of potentially inaccurate models based on erroneous or incomplete face knowledge and instead places the emphasis on the training examples from which the system which learn to distinguish a face. Most image-based approaches apply a window scanning technique for detecting faces, which due to it's exhaustive nature, increases computational demand.
Hjelmås and Low [1] divide the group of image-based approaches into three sections: Linear subspace methods, neural networks, and statistical approaches.

Linear Subspace methods

Images of human faces lie in a subspace of overall image space which can be represented by methods closely related to standard multivariate statistical analysis, including Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Factor Analysis (FA).
In the late 1980s, Sirovich and Kirby [17] developed a technique using PCA to represent human faces. The technique first finds the principal components of the

distribution of faces (expressed in terms of eigenvectors). Each face in the set can then be approximated by a linear combination of the largest eigenvectors, more commonly referred to as eigenfaces. Moghaddam and Pentland [18] later proposed a facial feature detector using 'Distance From Face Space' (DFFS – a measure of faceness), generated from eigenfeatures (eigeneyes, eigennose, eigenmouth), obtained from various facial feature templates in a training set. The performance of the eye locations was reported to be 94% with only 6% false positive (uniform background).

Various improvements to this simple system were made, notably that of Sung and Poggio [19] which represented the facespace by dividing it into subclasses. Their detector *(detailed in section 2.2.2)* involves four steps: image pre-processing, distribution-based model construction, image to model measurements taken, classification. One issue arising from the application of such systems is the problems associated with the collection of a representable set of training samples, particularly for non-face images. Sung and Pogio [19] suggest the use of a bootstrap algorithm, now widely implemented, which involves adding false detections to the training set as non-face examples.

Yang et al [20] proposed a method for face detection based on Factor Analysis (FA), which is similar to PCA but assumes the observed data samples come from a well defined model. Using a mixture of factor analysers, a set of training images is used to estimate the parameters in the mixture model. This model is then applied to sub-windows in the input image, and the probability of a face being present is returned. Yang et al also proposed a system using Kohonen's Self Organising Maps [21], and an LDA. Whereas PCA is aimed at representation, LDA aims for discrimination and is therefore well suited to face detection when the class of faces and non-faces is divided into subclasses.

Neural Networks

Early approaches based on the simple Multiple Layer Perceptrons (MLP) gave encouraging results on fairly simple datasets. The first advanced neural approach which reported performance statistics on a large, visually complex dataset, was by Rowley et al [22] *(detailed in section 2.2.1)*. Their system incorporates face knowledge in the retinally connected neural network architecture, with specialised window sizes designed to best capture facial information (e.g. horizontal strips to

identify the mouth). Images are pre-processed before being classified by the network, the output from which is post-processed to remove overlapping detections, resulting in one detection per face, and a reduction in false positives. Multiple networks were trained independently and their outputs combined using various arbitration methods to further improve performance. Their detector produced very impressive results, and has since been incorporated in many other systems including several commercial applications.

A different neural approach was suggested by Feraud et al [23] based on a Constrained Generative Model (CGM), an auto associative fully connected MLP with three layers. Lin et al [24], propose a fully automatic face recognition system based on probabilistic decision-based neural networks (PDBNN). For face detection the network has only one subnet representing the face class, and training is unsupervised.

Statistical Approaches

Systems based on information theory, support vector machines and Bayes' Decision Rule are examples of Image-Based approaches that do not fit into either of the other categories.

Colmenarez and Huang [25] developed a system based on Kullback relative information. This divergence is a non-negative measure of the difference between two probability density functions. During training, for each pair of pixels in the training set, a joint-histogram is used to create probability functions for the classes of faces and non-faces. The training procedure uses a large quantity of 11x11 pixel images and results in a set of look-up tables of likelihood ratios. Pairs of pixels which contribute poorly to detection are completely removed from the look-up tables to reduce computational requirements. The system was further improved by incorporating a bootstrap training algorithm.

In Osuna et al [26] a support vector machine (SVM) is applied to face detection. A SVM with a 2^{nd} degree polynomial as a kernel function is trained with a decomposition algorithm which guarantees global optimality. Images are pre processed and trained with a bootstrap learning algorithm *(more detail in section 2.2.6).* Other research into SVMs has attempted to improve the Osuna detector [26] with good results [27].

Schneiderman and Kanade [28, 29] propose two face detectors based on Bayes' decision rule.

$$\frac{P(\text{image} \mid \text{object})}{P(\text{image} \mid \text{non-object})} > \frac{P(\text{non-object})}{P(\text{object})}$$

A face exists at the current location if the above condition is true. The Bayes Decision rule is proven to give the optimal solution providing P(image | object) and P(image | non-object) are accurate. Schneiderman and Kanade [29] implement a wavelet transform in their second system which decomposes the image into 10 sub bands, from which 17 visual attributes are extracted, and treated as statistically independent random variables. The view based detector is capable of detecting frontal and profile views as it is trained with frontal and right profile images (which are mirrored to detect left profile faces). Promising results are report from these systems.

2.2 Examples of various face detection systems

This section details six examples of the various different approaches to the face detection problem, with examples of both feature based, and image based approaches. Performance statistics have been presented where available, although due to the lack of a common test set and procedure, comparison of the detectors is not provided.

2.2.1 Neural Network Approach: Rowley et al [22]

Henry A. Rowley, Shumeet Baluja, and Takeo Kanade [22] co-wrote an article presenting their Neural Network-Based Face Detection system. Their system arbitrates between multiple networks to improve performance over a single neural network. Capable of correctly locating upright frontal faces in GRAY-level images, the detector proved to perform well with varying lighting conditions, and relatively complex backgrounds. Presented here is a summary of the detector, its unique features, and an evaluation of its performance.

Images presented to the network go through a pre-processing step to improve the classification of the system. An attempt is made to equalise the intensity values of the pixels present in the examination window and histogram equalisation is performed to expand the range of intensities, in order to compensate for differences in lighting and camera input gains. The image is passed through a neural network (described in more detail in section XYZ), consisting of a layer of hidden units: four looking at 10x10 pixel sub regions, sixteen looking at 5x5 pixel sub regions (for detecting the eyes and nose), and six looking at overlapping 20x5 horizontal strips of pixels (for detecting the mouth). The output of the network is a single real value, indicating the presence of a face. The network is training with both faces and non-faces example images, such that it can learn to classify images in the relevant class correctly. A bootstrap method is used to simplify training, reducing the size of the training set, and increasing the speed of the training process, by adding false detections into the set as training progresses. To give the classifier invariance to small degrees of scaling, translation and rotation, the training set is extended through the addition of rotated (up to 10º), scaled (90%-110%) and translated (up to half a pixel) versions of the training images. The network is initialised with random weights, and is trained to output ‘1’ for face examples, and ‘-1’ for non-face examples.

Experiments with a single network give good detection rates (90.9% to 92.5%) but suffer large numbers of false positives. To improve the performance both merging overlapping detections from a single network and arbitrating among multiple networks can be implemented. Merging overlapping detections preserves those locations in an image with a high number of detections (within a neighbourhood), and eliminates locations with fewer detections. There are relatively few cases in which this heuristic fails, except in examples where one face partially occludes another. When using multiple networks, arbitrating between the networks in order to produce a signal output is necessary. With 'ANDing', the detection rate of faces is reduced (77.9%) as both networks had to detect a face for the overall output to be '1', but similarly very few false positives occur. When replaced with ORing, the detection rate improves (90.3%), but at the expense of a larger number of false positives. A third more complex method of arbitration, is to use a further stage of computation in the form of another Neural network. This method produces good detection rates (83.4% - 84.4%), with an acceptable number of false positives.

This approach to face detection is highly comparable to other studies in the field, able to detect between 77.9% and 90.3% of faces in a set of 130 test images, with an acceptable number of false detections. The system does however, have some limitations. It can only successfully detect upright frontal faces (looking directly at the camera), although this could be corrected by training a set of arbitrated multiple networks with a different training set.

2.2.2 Example Based Learning: Sung and Pogio [19]

Kah-Kay Sung and Tomaso Poggio [19] published a paper in 1998 entitled 'Example-based Learning for View-Based Human Face Detection'. It presents an example-based learning approach for locating unoccluded vertical frontal views of human faces at various scales in complex scenes, under a relatively wide range of lighting conditions. The performance of this detection system relies heavily on the face model. An image set of approximately 1000 mugshots is used to construct a distribution based generic face model, thus creating a face model thought to be more accurate and more descriptive, than manually synthesised models which rely on prior knowledge of facial structures. Once a model has been created, a decision procedure is then trained on a large set of "face" and "non-face" examples, to empirically discover a set of operating parameters and thresholds that separate the classes. Most similar approaches arrive at these conditions by manually examining a

few trial cases, whereas this automatic approach presented here is thought to provide statistically more reliable values, and is likely to be capable of learning any more complex high-dimensional or non-linear relationships for distinguishing between “face” and “non-face” patterns.

The system divides an image into multiple, possibly overlapping sub-images (19x19 pixels in size) and attempts to classify each image window as either “a face” or “not a face”. Pertinent information is extracted by taking measurements on each image window, which a pre-defined decision procedure uses to determine whether the enclosed pattern resembles the canonical face structure represented by the model. Multiple scales are handled by examining and classifying fixed sized window patterns on scaled versions of the image. Whenever a window “matches” the model, the system reports a face has been found and returns the location and scale of that face. The most critical part of this system is the algorithm for classifying window patterns. Input windows are masked to eliminate near boundary pixels which are likely to fall outside the spatial boundary of a face. A best fit brightness plane is subtracted from the image, and histogram equalisation is performed to correct for variations in illumination brightness and camera response curves. The model with which image windows are compared consists of a set of 12 prototypes, constructed by performing a modified version of the k-means clustering algorithm on a representative set of face images and non face images independently. When a new image is presented to the detector, a set of measurements such that each set of measurements is a vector of 12 distances from the new window pattern to each of the window pattern prototypes. Rather than using simple Euclidean distances, a new “Mahalanobis-like” distance metric was defined, which tests has proved performs very well. The computed distances are then fed into a Multi-Layer Perceptron (MLP) Neural Network with 12 pairs of input terminals, 24 hidden units and one output, and uses sigmoidal thresholding (see section xyz for more details). The network is trained using a standard back propagation algorithm to output a ‘1’ for a face pattern and ‘0’ otherwise. The training set consisted of 47316 “face” and “non-face” window patterns artificially enlarged to include slightly rotated and mirrored versions of the original face patterns.

The detector was shown to correctly identify 96.3% of all face patterns with 3 false detects with a database thought to represent the systems “best case” performance (high quality images with a fair amount a lighting variation). Those images that were

missed had either strong illumination shadows or large rotations. The average case performance was determined using a database consisting of varying quality images with complex backgrounds. The detector could correctly identify 79.9% of all face patterns with 5 false detects, with low quality images or hand drawn faces being missed, thought to be acceptable. Experiments were also done to evaluate the various parts of the system, which showed that the chosen distance metric outperforms other comparable metrics although the chosen MLP is often outperformed by a single Perceptron network suggesting the two classes are linearly separable. As with all model based approaches, the limitations lie primarily with the quality of the model. As models are fairly rigid, it has been suggested that this system could be duplicated, having several identical detectors running in parallel, each with distinctly different models for identifying different poses.

2.2.3 Decision Trees: Huang et al [30]

Jeffery Huang, Srinivas Gutta, and Harry Wechsler [30] proposed a novel algorithm for face detection using decision trees and shows its generality and feasibility using a database consisting of 2,340 face images from the FERET database.

The basic aim of all of these face detection systems is to separate the foreground face(object), from the background (clutter). This is done by constructing a set of rules for classifying objects given a training set of objects whose target class labels are known. The rules used for this detector form a decision tree (DT) which is derived using C4.5 [31]. C4.5 iteratively determines the facial feature most discriminatory, and splits the data into classes categorised by this feature. The next significant feature is then used to further partition each subset, and so on until each subset contains only one feature. This decision tree contains nodes which represent feature discrimination tests and associated exit branches representing those subclasses that satisfy the test. Once a tree has been constructed, a tree pruning mechanism is often invoked, discarding some of the unimportant sub trees whilst retaining those covering the largest number of examples, thus providing a more general description of the learned concept. Prior to images being presented to the classifier, they are pre-processed to focus computational resources and thus provide speed improvements. This initial stage comprises of three functions: histogram equalisation to correct illumination differences, edge detection to find transitions in pixel intensity and analysis of projection profiles ultimately yielding a boundary box.

A cropping stage then takes each 8x8 window identified, and labels it "face" or "non-face" using decision trees induced using both positive (face) and negative (non-face) examples. This training set contains 12 images whose resolution is 256x384 and consists of 2759 windows tagged CORRECT (positive examples), and 15673 tagged INCORRECT (negative examples). Finally the labelled output is post-processed to decide if a face is present by counting the number of face labels. If this value exceeds a threshold, this indicates the presence of a face. If found, the face boxes are then normalised to yield a uniform size of output image.

The accuracy achieved with this detector is quoted to be 96%, where accuracy information is gathered manually, based on visual observation that the face box includes both the eyes, nose, and mouth, and that the top of the face box is below the hairline. Experiments showed that the detector was capable of rejecting 23 out of 25 complex background scenery images (92%).

2.2.4 Model/Distance-Based: Kirchberg et al [32]

Klaus J. Kirchberg, Oliver Jesorsky, and Robert W. Frischolz [32] summarised their findings in a paper published in 2002 entitled "Genetic Model Optimisation for Hausdorff Distance-Based Face Localisation". They present a model-based approach to perform robust, high-speed face localisation based on the Hausdorff distance. An optimisation approach is proposed to create and improve an appropriate representative edge model by means of genetic algorithms. With model-based approaches, it is important to ensure that the chosen model accurately represents all face variations that the face detector is likely to be asked to classify. In the simplest form, this model could be an ellipse, loosely expressing this feature of the face, but this simple model is far from adequate for most applications. The model is optimised using a large database of sample images.

For simplicity this approach focuses on finding a single face (although extension to multiple faces in thought to be straight forward). An edge magnitude image is first calculated with the Sobel operator, where the relevant edge feature points are extracted by a locally adaptive threshold filter to compensate for variable illumination. It is assumed that this will produce a characteristic arrangement of segmentation points which allows the GRAY-level image to be presented in binary form. The associated face model used itself consists of a set of feature points and can be

represented as a binary image, with the chosen feature points being similar to the typically observed patterns in the images' face area. The model is optimised using a Simple Genetic Algorithm (SGA), which requires genotype coding of the face model, a fitness function, and some additional parameters. The genotype coding assumes the face model is symmetric along the vertical axis, and thus only half the model is coded in a two-dimensional binary genome. The fitness function assigns a real-valued number to the model that reflects the performance of the face localisation algorithm with that model. The model is tested with a set of sample images, and the ratio of found faces to the overall number of faces in the set determines the fitness value. A face is said to be found if some distance measure between true position and found position is below a certain threshold. To detect a face the model is superimposed over the image (at several discrete positions) and the similarity between the model and the covered part of the image is calculated. A face is considered to exist at the location yielding the highest similarity. In order to determine the "similarity" between the two binary images, a modified Hausdorff distance metric is used.

As the model can be scaled to detect faces of different sizes, it is important for the model to be sufficiently detailed to be able to do so, whilst maintaining an accurate generic representation of the face. A 45 x 47 model grid proved to be a good trade-off. Tests were carried out to analyse the performance on the Genetic Algorithms used. The GA was shown to perform better when developing a face model from scratch, with localisation performance of over 90% (compared to roughly 60% for hand drawn models). Face localisation could be further improved by using a multi-step detection approach using more than one model in different grades of detail. Each model would be optimised separately producing more exact face coordinates.

2.2.5 Rejection-Based Classification: Elad et al [33]

In 2002, Michael Elad, Yacov Hel-Or and Renato Keshet [33] co-wrote a paper presenting their Rejection-based classifier for face detection. Their classification algorithm exploits the property that a greater proportion of image windows will be classified as clutter (background or non-face elements), than as targets (face elements). This yields an effective yet low complexity target (face) detection algorithm, entitled the Maximal Rejection Classifier (MRC), which is based on linear successive rejection operations.

The face detection can be characterised, in practice, as designing a classifier C(z), which, given an input vector ‘z’ has to decide whether z belongs to the target class X or the Clutter class Y, i.e. decide whether the image contains a face or not. As the classifier lies at the heart of the face detection system, and is applied many times, simplifying it implies an efficient detection algorithm. The MRC maintains the simplicity of linear classifiers, but is powerful enough to deal with classes that are not linearly separable providing the clutter class and the convex hull of the target class are disjoint (a condition known as convex separability). The Classifier exploits the property of high clutter probability by ensuring clutter labelling is very fast, even at the expense of slower target labelling.

The MRC is an iterative algorithm that attempts to maximise the number of rejected clutter samples. Many samples are labelled as clutter in the early iterations, and thus are exempt from processing in successive iterations. When only a few non-clutter samples remain, these are labelled targets. The algorithm was demonstrated in this system, designed to detect frontal and vertical faces. A comprehensive face set was constructed containing 400 frontal and vertical face images 15x15 pixels in size. The non-face set was required to be must larger and was constructed by decomposing 54 arbitrary images into a Gaussian pyramid creating 1290 images. Each 15x15 block in these images was treated as a separate non-face example, effectively forming over 40 million non-face examples.

In order to improve the detection algorithm, various pre-processing techniques were presented; removal of the mean, shaping the frequency representation, masking out pixels. The performance was evaluated with the mean removal and masking pre-processing steps in place. All of six examples tested with the system required an effective number of close to two convolutions per each pixel in each resolution layer, with few false alarms. The results stated that the MRC face detection algorithm had performance accuracy comparable to the Rowley et al. [22] detector (detailed here), but with computation speed an order of magnitude faster.

2.2.6 Support Vector Machines: Osuna et al [26]

Edgar Osuna, Robert Freund and Federico Girosi [26] investigated the application of Support Vector Machines (SVMs) in computer vision and presented their findings in

an article entitled "Training Support Vector Machines: an Application to Face Detection". SVM is a learning technique developed by V. Vapnik and his team (AT&T Bell Labs), and is a relatively new method for training polynomial, neural network, or Radial Basis Functions (RBF) classifiers. This short summary contains a brief overview of this technique and provides an indication of its worth in the field in terms of performance statistics.

The proposed face detector is capable of detecting vertically oriented, unoccluded, frontal views of human faces in GRAY level images. Although scale variations and lighting conditions are dealt with to an extent by the detection system, there is no mention of the background complexity that can be effectively classified. This system requires no prior knowledge of faces, unlike both the Rowley et al approach which uses various feature specific receptors and the Sung and Pogio [19] approach which uses a complex face model. Decision surfaces are found through training equivalent to solving a linearly constrained quadratic programming problem which is challenging as the memory requirements grow as the square of the number of data points (the main limitation of this approach). A complex decision boundary is calculated to separate the two classes (face and non-face) with the largest possible margin, and fewest misclassifications. Those data points that play a vital role in this task usually lie close to the border and are called support vectors. An initial estimate of the decision boundary is made and then improved iteratively by computing the error (the number of support vectors that violate optimality). The face detection system is trained using a set of 19x19 images containing faces and non face patterns with associated expected outputs of '1' and '-1' respectively. Bootstrapping is used to strengthen the system; that is to say misclassified nonface images are stored for use as negative examples. Once trained, the system can classify new examples presented to it. Images are first rescaled several times and cut into 19x19 windows. Each of these 19x19 windows is then pre-processed and classified by the SVM. Windows identified as containing faces are outlined with a rectangle to indicate its location.

The results of the SVM are comparable with those of the systems proposed by Rowley et al [22], and Sung and Pogio [19]. Two tests set were used to analyse the performance of the system. The first, 'Set A' gave an impressive detection rate of 97.1% with only 4 false alarms. 'Set B', with more complex backgrounds and varying lighting conditions gave a 74.2% detection rate with 20 false alarms. This approach

is thought to extend well to other object detection problems, though with the current limitation of memory requirements.

2.3 Conclusion

The Literature survey describes some of the different approaches to face detection, and should give the reader an insight into some of the techniques being applied in the field. Feature-based and image-based approaches are discussed and both have a great deal to offer the face detection problem. Feature-based systems, are better suited to real-time application where the limitations of the Image-Based detectors restrict their use. Image-based systems have been shown to be more robust and more powerful, and have many uses. Despite reports of immense computational expense, recent attempts to optimise algorithms, coupled with advances in computer hardware, have reduced processing time dramatically.

Face detection is still most commonly applied as a pre-processing stage in face recognition systems. It has many potential applications in a wide range of fields, some of which are described here. Accompanied by voice data and fingerprints, face recognition is often an integral part of Biometric Identification systems, with model-based [34], template-based [35] and eigenface [36] methods all in development. Popularity with video conferencing, a technology used extensively for worldwide corporate communication, has driven research for improvements in the area. The integration of face detection, allows a single camera to track a speaker, or a multiple camera set-up [37], to choose which camera to use based upon the orientation of the speaker's face, improving the experience for those participating. The increasing popularity of digital cameras and internet improvements, have spurred a growth in the amount and accessibility of digital information. Efforts have been made to catalogue this digital content for search and retrieval purposes, including 'Name-It' [38] and the 'Informedia project' [39], both of which utilise the face detection system of Rowley et al [22], as does Webseer [40], as part of an image based search engine.

The survey details many different approaches, although, a comparative analysis of the techniques is not possible, due to the absence of a standard test procedure. The reported results in each case are from tests carried out on different test sets under different conditions, making performance comparison an impossible task.

3. Neural Network-Based Face Detection

As discussed in the Literature Survey, there are many different approaches to face detection, each with their own relative merits and limitations. One such approach is that of Neural Networks. This section gives a brief introduction to the theory of neural networks and presents a neural network-based face detector developed by Sanner (REF), with the aim of implementing and analysing the Rowley et al [22] detector with enhancements proposed by Sung and Poggio [19]. An explanation of Sanner's detector is given, and details of the experimental work carried out are also included.

3.1 Neural Network Theory

Neural Nets are essentially networks of simple neural processors, arranged and interconnected in parallel. Neural Networks are based on our current level of knowledge of the human brain, and attract interest from both engineers, who can use Neural Nets to solve a wide range of problems, and scientists who can use them to help further our understanding of the human brain. Since the early stages of development in the 1970's, interest in neural networks has spread through many fields, due to the speed of processing and ability to solve complex problems. As with all techniques though, there are limitations. They can be slow for complex problems, are often susceptible to noise, and can be too dependent on the training set used, but these effects can be minimised through careful design.

Neural Nets can be used to construct systems that are able to classify data into a given set or class, in the case of face detection, a set of images containing one or more face, and a set of images that contains no faces. Neural Networks consist of parallel interconnections of simple neural processors. Figure 2 shows an example of a single neural processor, or neuron. Neurons have many weighted inputs, that is to say each input (p_1, p_2, p_3… p_m) has a related weighting (w_1, w_2, w_3… w_m) according to its importance. Each of these inputs is a scalar, representing the data. In the case of face detection, the shade of GRAY of each pixel could be presented to the neuron in parallel (thus for a 10x10 pixel image, there would be 100 input lines p1 to p100, with respective weightings w1 to w100, corresponding to the 100 pixels in the input image).

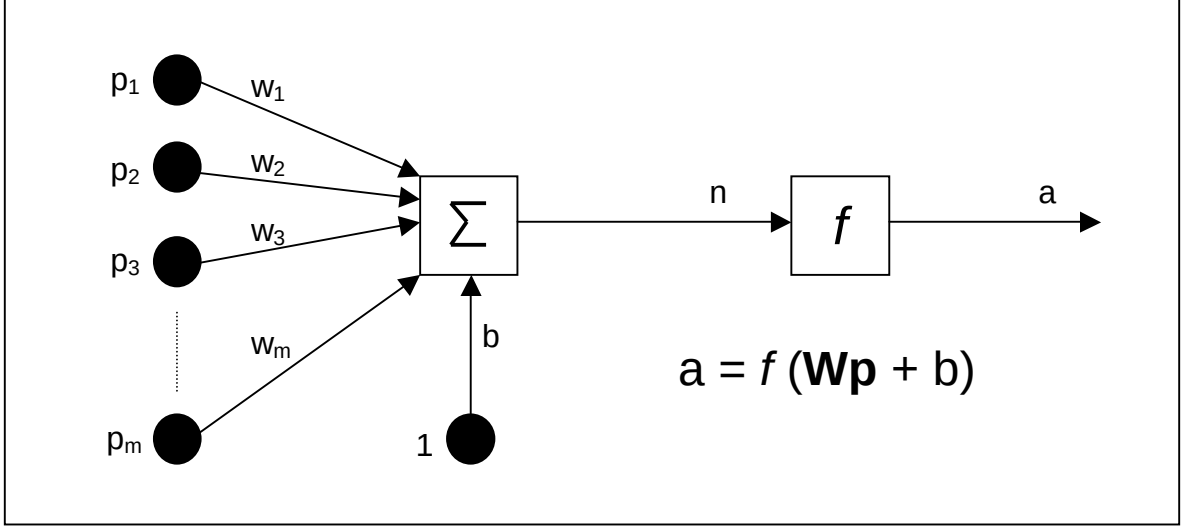


Figure 2 – a single neuron example neural network

The weighted inputs are combined together, and if present, a bias (b) is added. This is then passed as the argument to a transfer function (typically a pure linear, hard-limit, or log-sigmoid function), which outputs a value (a) representing the chosen classification.

Problems that are more complex can be realised by adding more neurons, forming multiple layers of several neurons, interconnected via a weighted matrix (*as shown in figure 3*). Additional layers of neurons not connected directly to the inputs or the outputs are called hidden layers (layers 1 and 2 in figure 3).

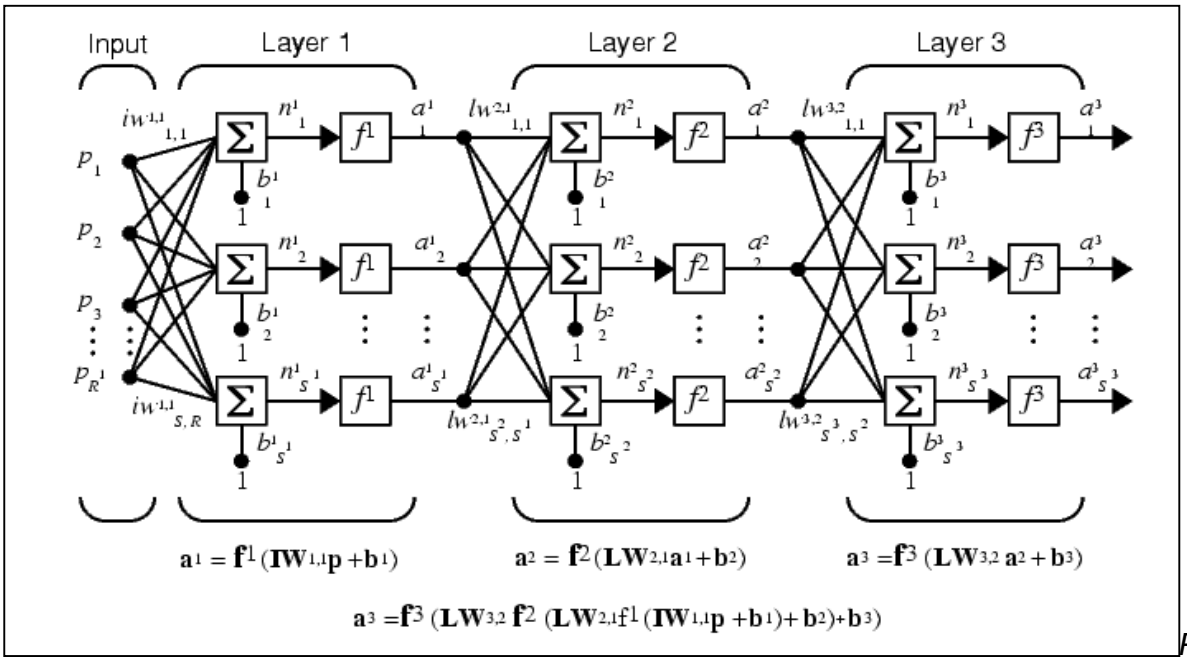


Figure 3 – An example of a three layer network with multiple neurons per layer taken from Matlab documentation

Once the architecture is established, the network must be trained. A labelled representative set of examples from each class is presented to the network, which attempts to classify each example. The weights and biases are initialised with small random values and updated incrementally, such that the performance of the detector improves producing a more accurate decision boundary for the problem.

Once trained, the network can be used to classify previously unseen images, indicating whether they contain faces or not, based-on the ‘location’ of the input relative to the decision boundary formed during training.

3.2 Neural Network-Based Detector Implementation

This detector was developed by Scott Sanner [41], and was designed to implement the Rowley-Baluja-Kanade neural face detector [22], with enhancements proposed by Sung and Poggio [19]. A description of the system is included to give the reader a detailed level of understanding of how the face detection problem can be solved. The face detector is analysed and the results are presented in the results section. Some minor improvements to the system are introduced here, and the system performance is re-analysed to provide a comparable set of results.

3.2.1 System Overview

The operation of the face detection system can be broken down into three main areas:

1. Initialisation (design and creation of a neural network)
2. Training (choice of training data, parameters, and training)
3. Classification (scanning images to locate faces)

A feedforward neural network is created which is trained using back propagation. The training set used contains examples of both face and non-face images, and the classifier is trained to output a value between 0.9 and -0.9 (0.9 firmly indicating the presence of a face, -0.9 firmly indicating the absence of a face). When a new image is presented to the network, the image is rescaled and divided into windows which are individually presented to the network for classification. Windows thought to contain a face are outlined with a black bounding box and on completion a copy of the image is displayed, indicating the locations of any faces detected. In the next section a more thorough description of the system is included detailing the operation of the detector.

3.2.2 System Description

Figure 4 shows the hierarchy of the Sanner Face Detector. There are two main functions: 'facetrain' to create and train a neural network and 'facescan' to scan new images for faces. A description of each function is given below.

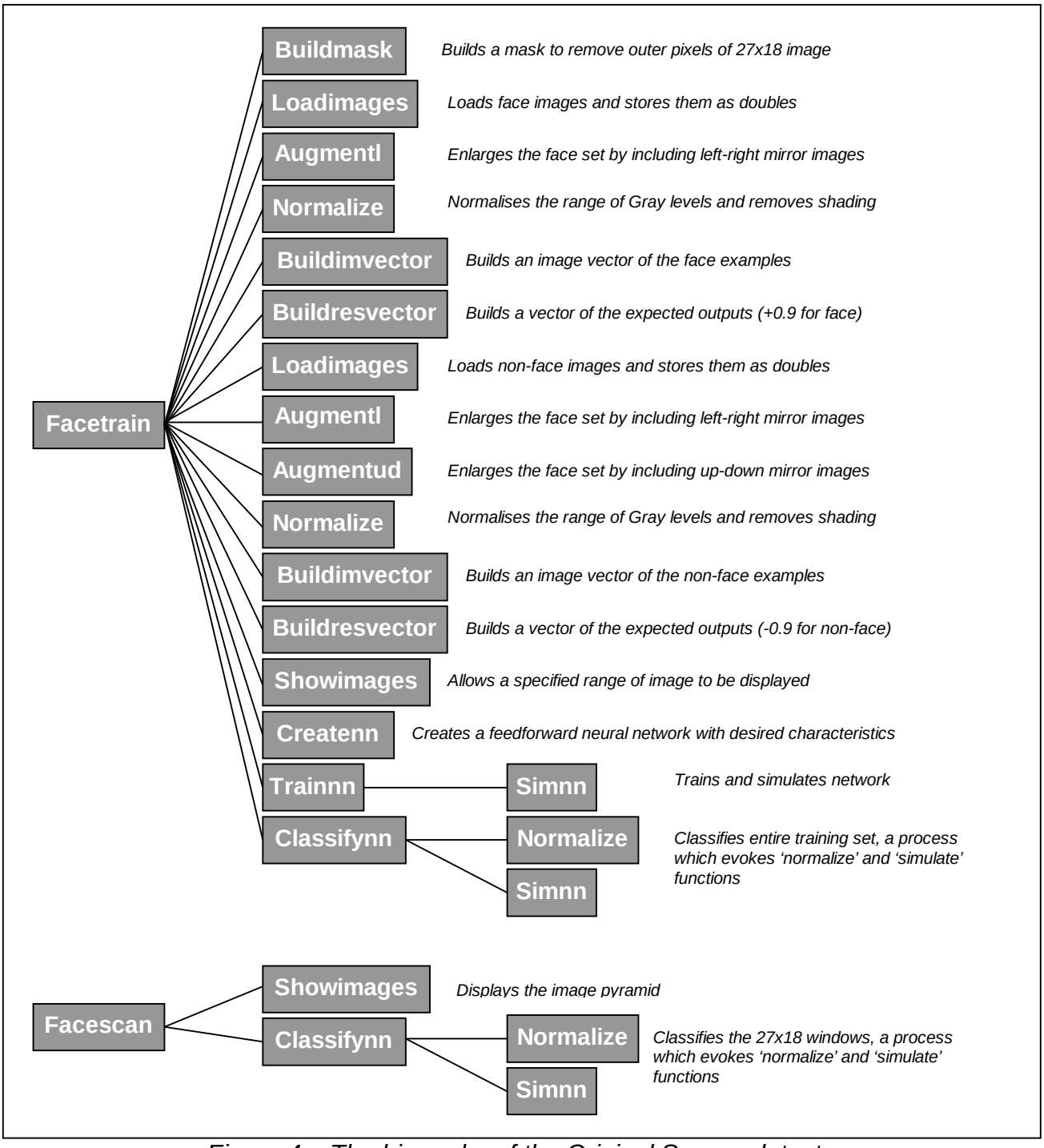


Figure 4 – The hierarchy of the Original Sanner detector

Facetrain

A set of 27x18 images from a training set is loaded and stored as an image vector. There are two vectors, one which contains numerous face examples, the other for non-face examples. Each image vector is then augmented, adding mirror-images of the original training examples, to create a larger training set. A mask is applied to the face examples, removing pixels outside of the oval mask to focus the attention of the classifier on the true face region. Pixels in the unmasked area are then normalised: a rough approximation of the shading plane is subtracted from the image to correct for single light source effects and the histogram is rescaled to ensure all images have the same gray level range (0-1). Once the training data has been pre-processed, the neural network is created (see figure 5). The network has 'NI' inputs, 25 hidden nodes, and just one output which indicates the presence, or absence of face. Each node's transfer function is of type 'tansig' – hyperbolic tangent sigmoid transfer function.

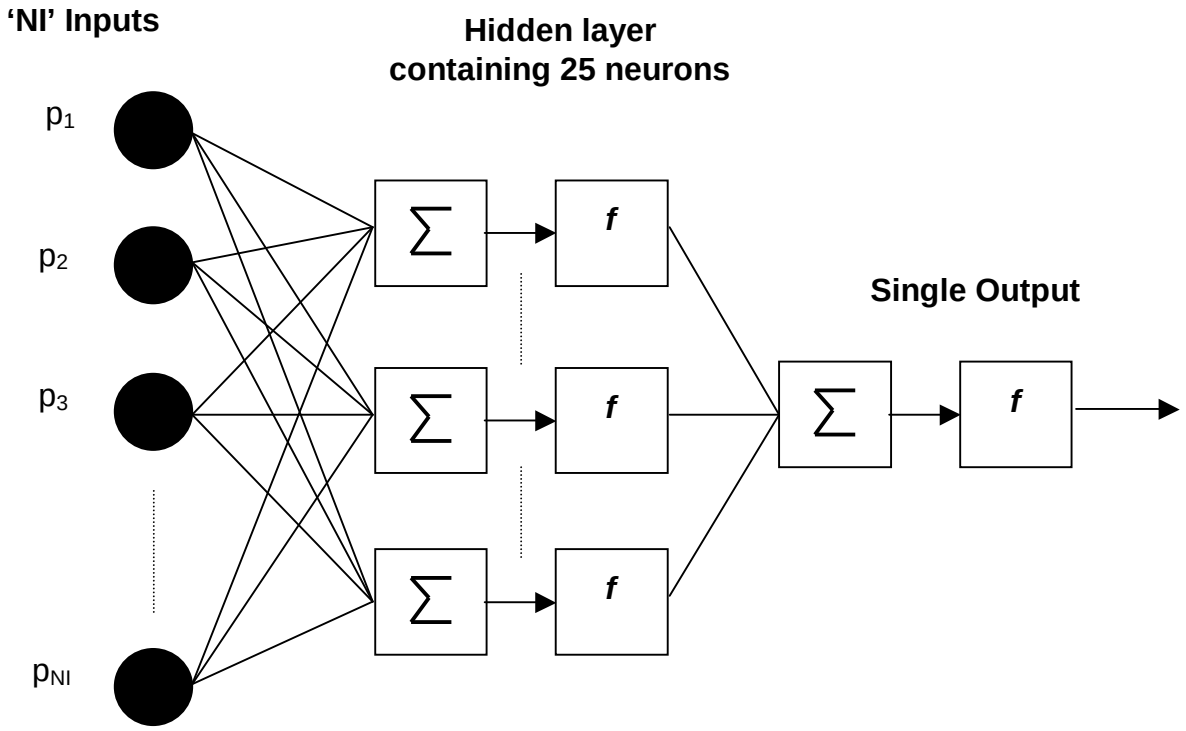


Figure 5 – A representation of the network architecture

Matlab's training algorithm 'traindm' is used which implements gradient decent back propagation with momentum. The network's weights and biases are updated according to gradient descent in order to improve the networks performance function. The Neural network is trained with all of the training data until convergence is achieved, or a decrease in performance is registered on the arbitrarily chosen validation set.

Facescan

Once the system has been created and trained, it is possible to classify new unseen images. The second function, “facescan”, conducts the final task, scanning previously unseen images for faces. Images are processed prior to classification, which involves the construction of an image resolution pyramid, and scanning 27x18 window regions, normalising each window before passing it to the network for classification. The image resolution pyramid is used to allow faces of differing scales (sizes) to be detected. When calling the ‘facescan’ function, a number of parameters can be specified which control the number of levels in the pyramid and the scale factor for resizing between levels, as well as other parameters specifying the network and mask to be used, and a threshold value, above which images are classified as faces.

3.2.3 Sanner’s Findings

This section focuses on Sanner’s reported findings, predominantly performance statistics and the limitations of the detector. Sanner completed several tests to investigate the choice of parameters, although a comprehensive analysis of true performance was not provided in the documentation, just a few examples of images classified by the system. Several strengths and limitations were identified. The image normalisation routine was identified as a strength, as it eases the collecting of examples and the submitting of faces for scanning, due to the degree of invariance to lighting conditions that it provides. It can also lead to improvements in computational efficiency, given the small size of the matrices used during normalisation. The detector was also able to correctly process a fairly wide range of poses, emotions, and lighting conditions, despite a relatively small and limited example training set. However, the detector was unable to detect rotated faces as there were no rotated face examples in the training set and the number of false positives (areas of background or scenery that the detector incorrectly identifies as faces), was unacceptable in some images. The implementation of a retinally connected network [22], was suggested to help reduce the effect of noise. The addition of a more comprehensive set of non-face examples in the training set was also suggested as a potential improvement, a task which is extremely difficult, although improved greatly through using a bootstrapping technique to construct the non-face example set *(see section 6 for more details).* The remainder of this report will analyse the Sanner [41] detector, evaluate its performance, and look at some possible improvements.

4. Experimental Work

This section details work carried out to measure the performance of the discussed Sanner face detector, and to analyse the improvements made.

4.1 Performance Analysis - Original Detector

The performance of the original face detector developed by Sanner will be discussed, and a set of optimal values for the various tuning parameters will be investigated. All the experimental work is to be carried out in Matlab using the existing code written by Scott Sanner. Some additional scripts will be written to implement any improvements, and to automate some of the performance testing experiments which would otherwise be a tedious repetitive procedure.

4.1.1 Classification Performance

At the very heart of the system lies the classifier, the object that actually makes the decision as to whether an image is receives contains a face or not. Initially tests were carried out to investigate how well the classifier could classify the data set on which it was trained. Although this is not indicative of true performance, it serves as a guide to how well the network is learning from the training data. The classify function written by Sanner, classifies 27x18 pixel images as either face or non-face images, producing a numerical value between -0.9 and 0.9 (0.9 strongly indicating the presence of a face, -0.9 indicating the absence of a face). The addition of a threshold value allows the classifier to be tuned somewhat, such that an image is marked as when the numerical output value exceeds the threshold. A script entitled “massclassify” makes use of the classify function by repeatedly classifying each example from the training data. For these experiments a virtual threshold of ‘0’ is assumed such that anything classified positive is defined as a face image, and negative values are defined as scenery images. This additional code is included in the appendices for reference. Face and non-face examples are classified separately. Whilst processing the face examples, correct classification values (exceeding the virtual threshold) are used to increment a ‘face counter’. Likewise a ‘non-face counter’ tallies the number of times non-face examples that are below the threshold value (again correct classification). Upon completion the classification rate and number of incorrect classifications for both the face and non-face examples are displayed. Figure 6 shows the results of ‘massclassify’ when used to classify the original training data set using Sanner’s original detector.

Number of Faces Examples:	30
Correct Classification Rate:	93.3%
Number of Incorrect Classifications:	2
Number of Scenery Examples:	40
Correct Classification Rate:	97.5%
Number of Incorrect Classifications:	1

Figure 6 – Original Face Detector Learning Performance Statistics

4.1.2 Parameter Optimisation

Several parameters in the system can be set to adjust various properties of the detector and tune its performance. Each of the parameters will be taken in turn, and experiments carried out to determine an optimal value for each one.

4.1.2.1 Mask

The mask is used to remove pixels towards the edge of the 27x18 images, thus focussing the attention of the network on the unmasked oval region, most likely to contain a face. The chosen mask is shown in the appendices and closely mirrors masks chosen by many others in the Literature survey. The unanimous acceptance of this mask throughout various approaches infers that the mask is somewhat optimal already, and thus no experiments will be done with other alternatives masks for the purposes of this project.

4.1.2.2 Network

Various characteristics of the network can be changed or varied including the network type, number of hidden nodes, training algorithm used and the training duration. Each parameter will be taken in turn and analysed.

Network Type

Changing the type of network used could potentially improve the performance of the detector, although the chosen feed-forward type is an excellent choice for this type of application, a choice which is mirrored by other neural network based face detection systems including Rowley et al [22]. Therefore due to the widespread acceptance of this network type, making changes at this stage is deemed unnecessary.

Number of hidden neurons

It is thought that any complexity of problem can be solved with just a single layer of hidden neurons. With a greater number of hidden neurons, there are more weights to tune during training, and thus a more complex a decision boundary can be formed (although too many neurons can lead to over fitting of the boundary to the training set, thus poor generalisation). The number of hidden neurons will be varied from 1 through to 1000 (25 being the default number in the original design), and the 'massclassify' function will be used to see how well the system learns the training set. Figure 7 shows the results of these experiments:

No. hidden neurons	Face detection rate %	Non-Face detection rate %	Total detection rate
1	99.2	97.5	98.1
10	100.0	99.0	99.4
20	98.3	100.0	99.4
25	99.2	99.5	99.4
30	99.2	98.5	98.8
40	100.0	100.0	100.0
50	100.0	100.0	100.0
100	100.0	100.0	100.0
1000	100.0	100.0	100.0

Figure 7 – The effect on performance of varying the number of hidden neurons

Training Algorithm

Various different training algorithms have been developed throughout the history of neural networks, each of which have their relative merits and limitations. Upon inspection the most relevant of the training algorithms that Matlab offers were 'trainrp' – resilient back propagation, 'trainscg' – conjugate gradient, and 'traingdm' – gradient decent with momentum. The network was designed to use traingdm, which is a good choice as this algorithm provides speed advantages, something which can be of great importance for an application such as this *(discussed in more detail in section 5)*. Although traingdm is likely to be the most optimal algorithm for this application, to confirm this, the 'massclassify' function was used to compare the detectors ability to learn the training set with the different training algorithms. Figure 8 shows the results from the massclassify function, in addition to the statistics regarding classification time, and errors displayed on completion of training.

Training Algorithm	**Trainlp**	**Trainscg**	**Traingdm**
F_{err} (Face err)	0.000	1.000	0.000
N_{err} (Non-face err)	1.000	0.074	0.010
T_{err} (Total err)	0.630	0.417	0.006
Face detection rate	100.0%	0%	100.0%
Non-face detection rate	0%	92.6%	99.0%
Total detection rate	37.0%	58.3%	99.4%
Classification time	3.00s	3.031s	2.969s

Figure 8 – Testing the detectors ability to classify the training data when trained with different training algorithms.

Training Duration

During training, the various weights and biases are updated incrementally upon processing the training data. Training continues until either the performance function reaches a specified goal, or until the number of iterations reaches a pre-defined maximum value. This maximum value was set as 500 for Sanner's detector [41], but has been provided as a parameter to the 'trainnn' function to allow this value to be adjusted. The maximum number of iterations will be varied from 1 to 1000, retraining the network for each value, and monitoring the performance statistics. The results are presented in Figure 9.

Number of Iterations	**F_{err}**	**N_{err}**	**T_{err}**	**F_{detect} (%)**	**N_{detect} (%)**	**T_{detect} (%)**
1	0.958	0.015	0.364	4.2	98.5	63.6
5	0.917	0.191	0.460	8.3	80.9	54.0
10	0.400	0.191	0.269	60.0	80.9	73.1
20	0.225	0.034	0.105	77.5	96.6	89.5
50	0.100	0.039	0.062	90.0	96.1	93.8
75	0.058	0.034	0.043	94.2	96.6	95.7
100	0.042	0.029	0.034	95.8	97.1	96.6
200	0.033	0.020	0.025	96.7	98.0	97.5
300	0.017	0.005	0.009	98.3	99.5	99.1
400	0.000	0.005	0.003	100.0	99.5	99.7
500	0.008	0.000	0.003	99.2	100.0	99.7
1000	0.000	0.000	0.000	100.0	100.0	100.0

Figure 9 – How varying the maximum number of iterations affects ability to learn training set.

Figure 9b shows how the error on the training set – reduces as the number of iterations is increased. Although figure 9 shows the training error falls to zero with 1000 iterations, this merely reflects that the network can perfectly classify the set on which it was trained, not a true indicator of real performance. As figure 9 and 9b show, the recommended number of iterations, 500, is a good choice.

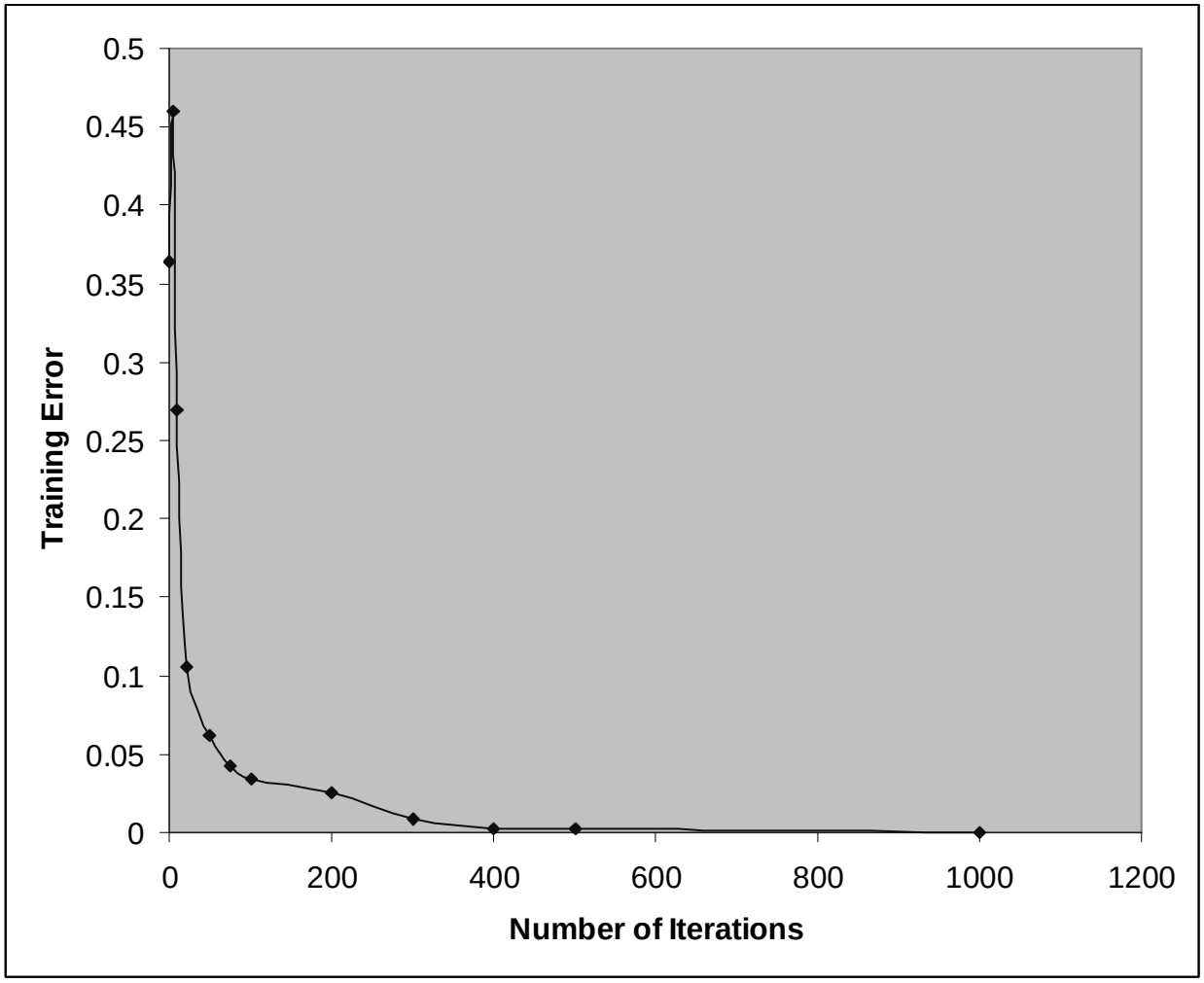


Figure 9b – The affect of increasing the number of iterations on training error

4.1.2.3 Threshold

Another parameter, the concept of which has been introduced previously, is a parameter that defines the numerical level above which an image must 'score' to be deemed a 'face'. This numerical score is the product of the classifier which produces a numerical output in the range -0.9 to 0.9 indicating the presence (0.9) or absence (-0.9) of a face. The threshold parameter is passed to the face scan function. When a new, previously unseen image is to be scanned with the function, a resolution pyramid is constructed, and each level is divided into 27x18 pixel windows. Each window is then classified individually, and those which produce a numerical value greater than the threshold are declared to be faces, and are outlined with a black bounding box indicating the location of the face. The threshold value therefore defines the selectivity of 'facescan' function. Sanner [41] suggests that a threshold value between 0.4 and 0.6 is optimal, a statement that should be confirmed by these findings. The results are tabulated in figure 10, showing how performance (detection rate relative to number of false positives) varies as the threshold is changed from 0 to 0.9. The following results were obtained by repeatedly scanning a test set containing 13 images – 8 face/5 non-face, taken from a dataset compiled by Rowley et al [22], at CMU [42]. A script, 'threshold', was written to automate this process (and is included in the appendices for reference), although the statistics were obtained through manual inspection of the bounding boxes (with the aid of the function 'showimage').

	No. faces	THR 0.0		THR 0.1		THR 0.2		THR 0.3		THR 0.4		THR 0.5		THR 0.6		THR 0.7		THR 0.8		THR 0.9	
		F_{detect}	Num_{fd}	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}
FF1	1	1	*60*	1	*50*	1	*45*	1	*35*	1	*25*	1	*20*	1	*15*	1	12	1	6	0	2
FF2	1	1	*60*	1	*55*	1	*50*	1	*45*	1	*40*	1	*30*	1	*20*	1	*10*	1	*3*	1	*0*
FF3	1	1	*60*	1	*55*	1	*45*	1	*20*	1	11	1	5	1	4	1	0	0	0	0	0
FF4	1	1	*55*	1	*45*	1	*40*	1	*30*	1	*20*	1	*12*	1	5	1	5	1	2	0	0
FF5	2	2	*55*	2	*50*	2	*40*	2	*30*	2	*20*	2	13	2	9	0	4	0	0	0	0
FF6	1	1	*60*	1	*50*	1	*45*	1	*40*	1	*30*	1	*25*	1	13	1	2	0	0	0	0
FF7	2	2	*60*	2	*55*	2	*45*	2	*40*	2	*30*	2	*20*	2	17	1	11	0	5	0	0
FF8	7	6	*70*	5	*60*	4	*50*	4	*40*	3	*25*	3	*20*	1	17	1	9	0	2	0	0
NF1	0	0	*200*	0	*170*	0	*150*	0	*120*	0	*100*	0	*80*	0	*60*	0	*50*	0	*45*	0	7
NF2	0	0	*250*	0	*200*	0	*170*	0	*140*	0	*100*	0	*80*	0	*50*	0	*35*	0	16	0	5
NF3	0	0	*550*	0	*500*	0	*450*	0	*390*	0	*330*	0	*280*	0	*200*	0	*120*	0	*70*	0	*40*
NF4	0	0	*600*	0	*520*	0	*470*	0	*410*	0	*360*	0	*270*	0	*110*	0	*60*	0	50	0	1
NF5	0	0	*450*	0	*400*	0	*330*	0	*260*	0	*200*	0	*170*	0	*110*	0	*55*	0	12	0	0

Figure 10 – A table of thresholding experiments with the original Sanner detector

The results from figure 10 are presented in graphical form in figure 10b. The plot highlights the trade off between correct detection rate and the number of false positives. From figures 10 and 10b a threshold of 0.7 has been found to be optimal, that is to say best detection rate relative to number of false positives.

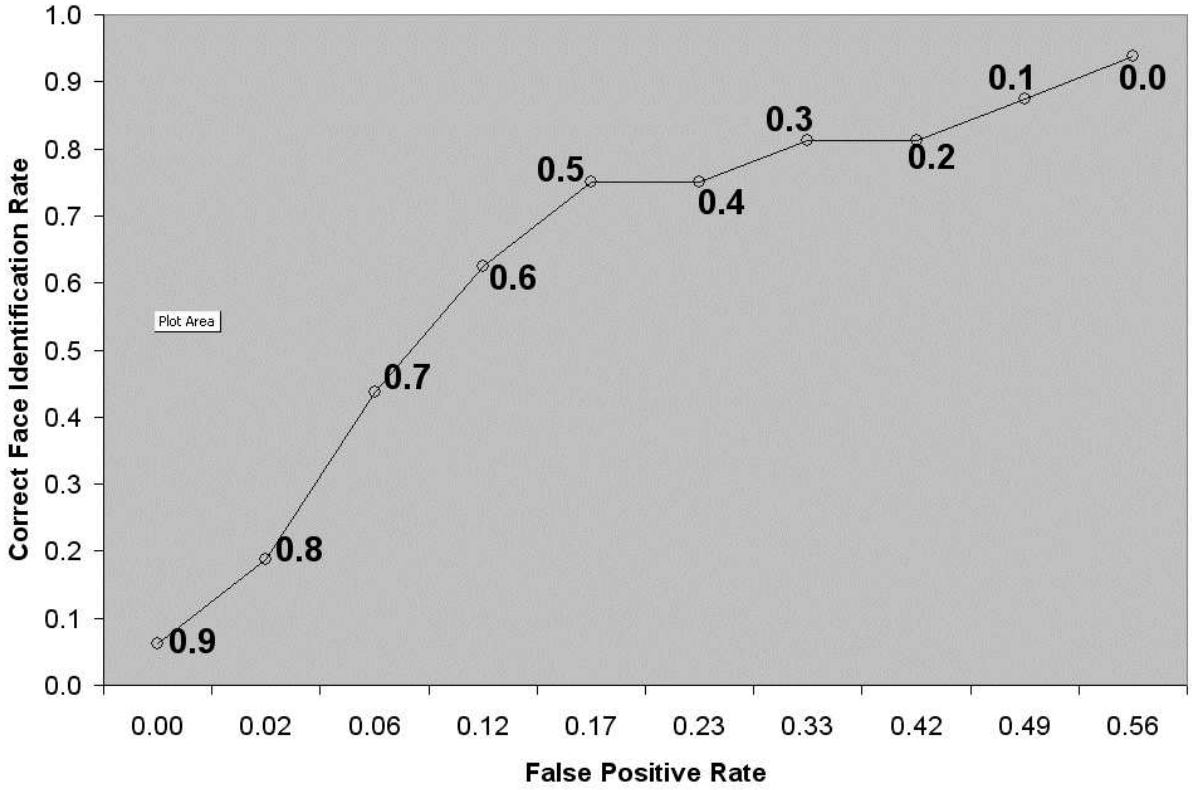


Figure 10b – An ROC Plot for the Original Face Detector

4.1.2.4 Pyramid

When images are scanned for faces (using the facescan function), an image pyramid is constructed. This pyramid, designed to facilitate the detection of faces at different scales, is formed of a discrete number of levels. The original image is resized by a pre-defined scale factor for each level of the pyramid. Sanner provided for the adjustment of three important parameters regarding the construction of the pyramid, each of which will be analysed in turn.

Number of Pyramid Levels

The greater the number of levels, the more faces at distance from the camera can be detected, at the expense of an increase in the number of false detections. With fewer levels, the scale of possible face detections will be reduced, but similarly the number of false positives is likely to decrease. Sanner [41] suggests that 6 levels for the pyramid is optimal, an assertion mirrored by others working in the field *(see section 2)*. To confirm this optimal value, two images have been created containing faces of different sizes. Both the images were constructed by combining a number of images collected by Rowley et al for their face detector [42]. Each of the images was scanned using the facescan function, with the "Levels" parameter varied between 1 and 20. The results are shown in figure 11.

No. Levels	SCALE01 (6 faces)		SCALE02 (31 faces)	
	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}
1	0	9	1	18
2	2	16	8	26
4	4	30	18	33
6	4	37	20	48
8	4	37	22	51
10	4	37	22	54
12	4	39	27	67
14	4	40	27	69
16	4	42	27	73
18	4	45	27	76
20	4	49	27	81

Figure 11 – The affect that varying the number of pyramid levels has on classification performance (face detection rate F_{detect}, no. false positives Num_{fd})

Pyramid Scale-Factor

The original image is resized to form the various layers of the resolution pyramid. Each level is resized by a specified factor of the previous level, thus this 'scale factor' also influences the range of scales of faces that can be detected. Sanner [41] suggested an optimal value of 1.2 what appears to be a popular choice throughout the literature. The following tests are expected to agree with this value. The same two images were scanned using once more, this time varying the scale factor from 1.1 to 1.5 at 0.1 increments. All other parameters were kept constant. The results can be seen in figure 12.

Scale Factor	SCALE01 (6 faces)		SCALE02 (31 faces)	
	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}
1.1	4	64	19	36
1.2	4	46	24	34
1.3	3	37	24	31
1.4	3	25	23	28
1.5	3	25	22	26

Figure 12 – The affect of varying the scale factor on classification performance (face detection rate F_{detect}, no. false positives Num_{fd})

Pyramid Start-Level

The start level governs at which level of the pyramid facescan should begin. This parameter is set to 1 by default, such that all levels are scanned. The start level can be adjusted to remove detections of faces below a certain size (and thus reduce false positives). A ten level pyramid constructed with a scale factor of 1.2 was used for the following experiments, during which the start parameter was varied from 1 through to 9. The results can be seen in figure 13.

Start Level	SCALE01 (6 faces)		SCALE02 (31 faces)	
	F_{detect}	Num_{fd}	F_{detect}	Num_{fd}
1	4	40	23	41
2	4	32	23	34
3	3	24	22	32
4	3	17	20	25
5	2	12	6	20
6	0	7	4	14
7	0	4	4	12
8	0	3	4	9
9	0	1	4	8
10	0	0	1	4

Figure 13 – The affect of varying the start level on classification performance

4.1.3 Performance Evaluation

In Section 4.1.1, a face detection rate of 93.3% with only one misclassified scenery image *(see figure 6)* was reported. These statistics only indicate how well the network is able to classify the training data, and thus how well it has "learned" the training set during training. An insight into the true performance of the system can only be achieved through the use of an independent test set of "unseen" images, producing statistics more indicative of system performance in real world applications. No standard test set or test procedure exists, and thus comparison between this system, and other proposed face detection systems is almost impossible, an observation made in the Literature Survey. In order to produce a realistic reflection of performance in the real world, a test set should evaluate the performance of the detector under a wide range of conditions, representing the true nature of the variation fraught environment. The test set should contain images of various resolutions, with both face and scenery examples present. Faces should be present at numerous scales, under various lighting conditions, and with differing complexities of background. The majority of faces should be frontal faces (as this detector is designed to detect frontal faces), although slight rotations can be introduced. Two test sets have been chosen to evaluate the performance of Sanner's Face detector [41]. The first, containing 42 face images and 0 scenery images, is thought to be easier than the second test set containing 52 face images, and 13 fairly complex scenery images. Both test sets were obtained from CMU [42], and contain images used in the evaluation of the Rowley et al [22] detector (a summary of the images in the two test sets is included in section 9). Two scripts ('testset01' and 'testset02') have been written to automate the scanning of the training images. Each image is scanned individually using a pyramid (levels = 6, scale-factor = 1.2). Bounding boxes are drawn around image windows which exceed the threshold (0.8) and a resultant image is returned and stored for manual inspection *(see section 9 for code)*. Figure 14 shows the performance of Sanner's original detector [41] when analysed with each of the two test sets. The number of images in these test sets is ambiguous; it depends on whether you count partially occluded faces, profile faces, illustrations of faces, and animal faces. For the purposes of this project, 'testset01' contains 178, and 'testset02' contains 180. The results show that the original detector detected 29% of the faces in 'testset01' with an average of 11 false positives per image, and 44% of the faces in 'testset02' with an average of 18 false positives.

Test Set 01			
Image Number	**No. Faces In Image**	**No. Faces Detected**	**No. False Positives**
1	5	2	7
2	1	0	12
3	4	3	18
4	2	0	8
5	2	0	**100**
6	2	0	0
7	2	1	2
8	1	0	1
9	1	1	1
10	1	0	6
11	1	0	1
12	1	0	1
13	1	0	1
14	8	3	11
15	1	0	3
16	25	8	30
17	1	0	2
18	4	0	7
19	5	2	8
20	1	0	1
21	1	0	2
22	2	0	0
23	1	0	12
24	1	0	1
25	1	0	0
26	2	0	3
27	1	0	2
28	3	1	0
29	2	0	1
30	2	0	0
31	8	3	19
32	8	4	6
33	8	7	8
34	7	0	4
35	8	2	17
36	12	3	24
37	4	1	**80**
38	14	6	12
39	1	1	12
40	1	1	2
41	9	1	17
42	9	1	9

Test Set 02			
Image Number	**No. Faces In Image**	**No. Faces Detected**	**No. False Positives**
1	7	2	**50**
2	1	0	**60**
3	1	0	2
4	1	0	0
5	1	1	2
6	1	0	1
7	1	0	1
8	6	1	6
9	1	0	0
10	1	0	0
11	0	0	20
12	1	0	10
13	57	34	37
14	0	0	16
15	1	0	0
16	1	0	10
17	1	1	**70**
18	6	4	7
19	2	0	1
20	6	3	6
21	0	0	3
22	0	0	39
23	1	0	4
24	2	0	0
25	0	0	30
26	1	0	9
27	1	0	**40**
28	1	0	3
29	1	0	12
30	2	0	**25**
31	9	3	5
32	4	4	13
33	1	0	**50**
34	0	0	**35**
35	2	1	15
36	3	0	2
37	1	0	**30**
38	1	0	1
39	1	0	4
40	14	10	4
41	8	1	18
42	2	2	**100**
43	1	0	1
44	0	0	14
45	3	3	**50**
46	7	4	4
47	1	0	0
48	1	0	7
49	4	3	6
50	1	1	1
51	0	0	8
52	1	0	1
53	1	0	5
54	1	0	14
55	0	0	**80**
56	1	0	14
57	1	1	1
58	0	0	14
59	0	0	21
60	0	0	2
61	2	0	4
62	2	1	4
63	1	0	2
64	0	0	**100**
65	1	0	0

Performance Analysis for Original Detector Configuration

Test Set 01

- Face Detection Rate: 29%
- Average Number of False Positives: 11

Test Set 02

- Face Detection Rate: 44%
- Average Number of False Positives: 18

NB: Numbers in bold (**50**) indicate an estimate of the number of detections. As the images are inspected manually, it with large numbers of overlapping detections, an accurate number is impossible to determine and thus an estimate is suggested.

Figure 14 – Original Face Detector Performance Evaluation

Sanner [41] identified some limitations with his face detector, mainly the number of false positives (windows the classifier incorrectly believes are faces). This infers a weakness in the training set. The detector was also stated to provide translational invariance, although rotation in faces was not catered for, with the failure to detect several rotated faces in the test sets. An attempt has been made to improve the detector to overcome some of these limitations.

4.2.1 Training Data Replacement

The first modification regards the training data. The training data was completely replaced with a set used to train a Face Detection System developed by Mottram [43]. This data set contains 111 training examples, 51 non-face and 60 face. The set was adjusted to comply with Sanner's Detector [41] by resizing the images to 27x18 pixels, converting each image to an 8 bit Grayscale PNG, and renaming them. Sanner's detector was then retrained with the new set using the same training parameters as were used originally. The script 'massclassify' was executed, as in section 4.1.1. 100% of the face examples and 98.5% of the non-face examples were classified correctly, inferring the decision boundary formed during training allows accurate division of face and non-face images. To determine the effect on performance, the performance evaluation procedure was repeated (as described in section 4.1.3, using the same test sets and parameters. The results are shown in figure 15.

Test Set 01			
Image Number	**No. Faces In Image**	**No. Faces Detected**	**No. False Positives**
1	5	5	**100**
2	1	0	18
3	4	0	**100**
4	2	0	**100**
5	2	1	**100**
6	2	1	**50**
7	2	0	24
8	1	0	26
9	1	1	**50**
10	1	1	30
11	1	0	24
12	1	1	**30**
13	1	1	16
14	8	1	15
15	1	0	26
16	25	22	**100**
17	1	1	11
18	4	0	**200**
19	5	4	**100**
20	1	0	28
21	1	0	**35**
22	2	0	36
23	1	0	**35**
24	1	0	12
25	1	0	4
26	2	0	9
27	1	0	13
28	3	3	18
29	2	0	**30**
30	2	0	10
31	8	5	**30**
32	8	4	10
33	8	2	14
34	7	0	19
35	8	1	**100**
36	12	6	**75**
37	4	3	**150**
38	14	2	**50**
39	1	1	**40**
40	1	1	**40**
41	9	1	**50**
42	9	6	**35**

Test Set 02			
Image Number	**No. Faces In Image**	**No. Faces Detected**	**No. False Positives**
1	7	2	**50**
2	1	0	**200**
3	1	0	**35**
4	1	0	**40**
5	1	1	**30**
6	1	0	**25**
7	1	0	**30**
8	6	3	**30**
9	1	0	**1**
10	1	1	4
11	0	0	15
12	1	1	7
13	57	47	**60**
14	0	0	34
15	1	0	**25**
16	1	0	**60**
17	1	1	**70**
18	6	5	**70**
19	2	2	22
20	6	2	23
21	0	0	8
22	0	0	**50**
23	1	0	29
24	2	2	10
25	0	0	**50**
26	1	0	17
27	1	0	**70**
28	1	0	2
29	1	0	27
30	2	1	**100**
31	9	2	51
32	4	3	**60**
33	1	0	**70**
34	0	0	**50**
35	2	2	**60**
36	3	0	42
37	1	0	**100**
38	1	0	42
39	1	0	68
40	14	11	33
41	8	2	**60**
42	2	2	**120**
43	1	0	17
44	0	0	24
45	3	3	**80**
46	7	1	**50**
47	1	0	46
48	1	0	**40**
49	4	4	38
50	1	1	15
51	0	0	27
52	1	1	29
53	1	0	6
54	1	1	6
55	0	0	**120**
56	1	0	**75**
57	1	0	**25**
58	0	0	**50**
59	0	0	53
60	0	0	9
61	2	1	32
62	2	1	28
63	1	0	8
64	0	0	**30**
65	1	0	2

Performance Analysis for Detector Trained with Replaced Training Set

Test Set 01

- Face Detection Rate: 44%
- Average Number of False Positives: 18

Test Set 02

- Face Detection Rate: 57%
- Average Number of False Positives: 46

NB: Numbers in bold (**50**) indicate an estimate of the number of detections. As the images are inspected manually, it with large numbers of overlapping detections, an accurate number is impossible to determine and thus an estimate is suggested.

Figure 15 –Face Detector Performance Evaluation with replaced training set

4.2.2 Further Enhancements

The modified set achieved higher detection rates than the original Sanner detector setup [41], at the expense of a greater number of false positives. In order to reduce the number of false positives, the training set must be strengthened with a more representative set of non-face examples. This final batch of enhancements implements some of these improvements.

The test scans showed that uniform blocks gray, often present background scenes, were consistently mistaken for faces in the original detector. Also linear boundaries between two different gray levels (edges) were also misclassified. In order to strengthen the detector in these areas, uniform images (plain white, gray, and black), random noise images, and images of the edges of various shapes will be included as non-face examples. Regarding face examples, rotated faces will be included. The lack of rotation invariance was reflected when both the original system (section 4.1.3), and the modified version (section 4.2.1) were used to scan the test sets. Rotated examples will be added in an attempt to provide some degree of rotation invariance, in an attempt to further improve the detection rate. Images of faces with spectacles will also be included as these were often missed by the detectors in the previous tests. Finally a noise function will be implemented to extend both the face and non-face example sets to include noisy versions, in an attempt to make the detector somewhat invariant to noise.

With the additional training data, there are 125 face images (resulting in 500 face examples after augmentation of the face image vector). In addition there are 117 non-face images (936 non-face examples after augmentation). Once again the network was used to scan the two test sets, the results for which are given in figure 16. The improvements made did decrease the number of false positives but only by an average of 4 per image, with the value still be unacceptable.

Test Set 01			
Image Number	**No. Faces In Image**	**No. Faces Detected**	**No. False Positives**
1	5	5	**75**
2	1	0	15
3	4	0	**80**
4	2	0	**80**
5	2	1	**100**
6	2	1	42
7	2	0	19
8	1	0	25
9	1	1	**40**
10	1	1	30
11	1	0	25
12	1	1	33
13	1	1	15
14	8	1	18
15	1	0	23
16	25	22	**100**
17	1	1	10
18	4	0	**150**
19	5	4	**100**
20	1	0	24
21	1	0	**40**
22	2	0	36
23	1	0	36
24	1	0	12
25	1	0	4
26	2	0	9
27	1	0	13
28	3	3	18
29	2	0	**30**
30	2	0	10
31	8	6	**50**
32	8	4	10
33	8	2	14
34	7	0	18
35	8	1	**80**
36	12	6	**75**
37	4	3	**120**
38	14	2	**50**
39	1	1	40
40	1	1	34
41	9	1	44
42	9	6	**35**

Test Set 02			
Image Number	**No. Faces In Image**	**No. Faces Detected**	**No. False Positives**
1	7	2	45
2	1	0	**200**
3	1	1	**30**
4	1	0	**40**
5	1	1	**30**
6	1	0	**25**
7	1	0	**30**
8	6	3	**30**
9	1	0	1
10	1	1	4
11	0	0	12
12	1	1	**7**
13	57	47	**70**
14	0	0	34
15	1	0	**35**
16	1	0	**50**
17	1	1	**60**
18	6	5	**60**
19	2	2	30
20	6	2	27
21	0	0	7
22	0	0	**50**
23	1	0	29
24	2	2	10
25	0	0	43
26	1	0	16
27	1	0	**70**
28	1	0	2
29	1	0	27
30	2	1	**80**
31	9	2	44
32	4	3	**60**
33	1	0	**60**
34	0	0	**50**
35	2	2	**60**
36	3	0	42
37	1	0	**80**
38	1	0	39
39	1	0	59
40	14	11	24
41	8	2	**60**
42	2	2	**100**
43	1	0	17
44	0	0	24
45	3	3	**80**
46	7	1	**50**
47	1	0	37
48	1	0	33
49	4	4	38
50	1	1	15
51	0	0	25
52	1	1	24
53	1	0	6
54	1	1	6
55	0	0	**100**
56	1	0	**75**
57	1	0	22
58	0	0	**40**
59	0	0	53
60	0	0	9
61	2	1	32
62	2	1	27
63	1	0	8
64	0	0	**30**
65	1	0	**2**

Performance Analysis for Detector Trained with Improved Training Set

Test Set 01

- Face Detection Rate: 43%
- Average Number of False Positives: 42

Test Set 02

- Face Detection Rate: 58%
- Average Number of False Positives: 44

NB: Numbers in bold (**50**) indicate an estimate of the number of detections. As the images are inspected manually, it with large numbers of overlapping detections, an accurate number is impossible to determine and thus an estimate is suggested.

Figure 16 - Face Detector Performance Evaluation with final improvements

5. Discussion

The above experiments were carried out to analyse the performance of the original Sanner detector [41], and the changes made to improve the performance. This section will describe the meaning of the results presented above and the relevance of the findings to this project and to potential applications.

5.1 Original Face Detection System

Sanner's face detector was designed with care and the chosen parameters suggested by Sanner [41] proved somewhat optimal in the tests carried out.

With reference to the network design, the type of network chosen is commonly used in other neural network solutions including the detector proposed by Rowley et al [22]. With the suggested 25 hidden nodes, good performance was seen. The addition of more nodes led to a negligible increase in performance with the adverse effect of increased computational time and the risk of possible over fitting. The chosen training algorithm is computationally faster than the other two on test and performs well provided the training set is comprehensive. Sanner [41] suggested a training duration of 500 iterations which was shown to be optimal in the experiments. The performance improved as the number of iterations was increased, although with more than 500 iterations, the error on the randomly chosen validation set remained constant, suggesting no improvement in performance (and again the increased risk of over fitting). Overall the network design seems well tuned to provide optimal performance, with minimal computational demands, whilst limiting the effects of possible over fitting.

With reference to the 'facescan' parameters, adjustable to tune the performance of the face detection procedure, the tests highlighted a threshold value of 0.5 was ideal for the limited test set. A higher threshold level was shown to minimise the number of false positives, although at the expense of a decrease in the number of face detections for threshold levels above 0.5. Sanner suggests a threshold between 0.4 and 0.6 which the experiments confirm. The recommended pyramid characteristics: (number of levels - 6 and scale factor - 1.2), were also somewhat verified by the initial experiments. The network performed well on the two test images at the suggested value of 6 levels, although in the case of the second image "SCALE02" it

was shown that by increasing the number of levels to 12 gave better performance ('best performance' is measured as the point at which the maximum numbers of faces are detected with the fewest number of false positives). Extensive parameter tweaking can therefore be used to improve the performance for an individual image, although approximately six levels proved adequate and is likely yield a more general solution. Similarly the suggested scale factor of 1.2 was seen to be optimal for both images tested, although a more comprehensive set of tests is required to ensure the worth of this and many of the other assertions. The start level, a parameter used to instruct the detector to ignore detections below a certain size, is set as 1 by default, resulting in the scanning of all pyramid levels. The two images scanned would have benefited from a starting level of 2 as fewer false positives were detected without a reduction in detection rate. The worth of this parameter depends very much on the image to be scanned, and thus if the application domain was known, this along with the other pyramid characteristics could be used to give significant performance gains.

An indication of the true performance of the detector was given in section 4.1.3 were the network was used to classify unseen images, essentially testing the generality of the class divide. The two test sets returned impressive results for such a simple system with so few training examples, reflecting the true nature and power of the neural networks approach. Although the detection rate was fairly low for both sets, especially 'testset01', the number of false positives was similarly relatively low. The faces that the system failed to detect were often rotated, bespectacled, occluded or suffered adverse lighting conditions, some of which were not catered for by the system or the training data *(a summary of the test images is given in section 9.3)*.

5.2 Implemented Improvements

There were two attempts to improve the system performance: the first entirely replaced the training data set, the second complemented the new set with additional image with the aim of improving the detection rate slightly, but mainly to reduce the number of false positives. The network was retrained with the new image datasets, and then used to scan the same two test sets as the original detection system.

	Original Detector	Replaced Dataset	Improved Dataset
Testset01 - F_{detect}	29%	43%	43%
Testset01 - Num_{fd}	11	47	42
Testset02 - F_{detect}	44%	57%	58%
Testset02 - Num_{fd}	18	46	44

Figure 17 – Comparison of performance statistics

The results (figure 17 and 17b) were encouraging with a substantial increase in detection rates, although an accompanying increase in the number of false detections. In the first instance, the training data was completely replaced resulting in many more faces being detected. The number of false positives indicated a clear problem with the training data, and thus improvements were made to the training set. Despite these attempts, the improvement was fairly small, as the number of additional images added to the training set was small in comparison to the size of the set, and thus contributed little to the overall "location" of the class decision boundary. This highlights the importance of the non-face examples, the difficultly in constructing a representative set, and the need for a better technique for doing so *(see section 6 and 7.1 for more details)*.

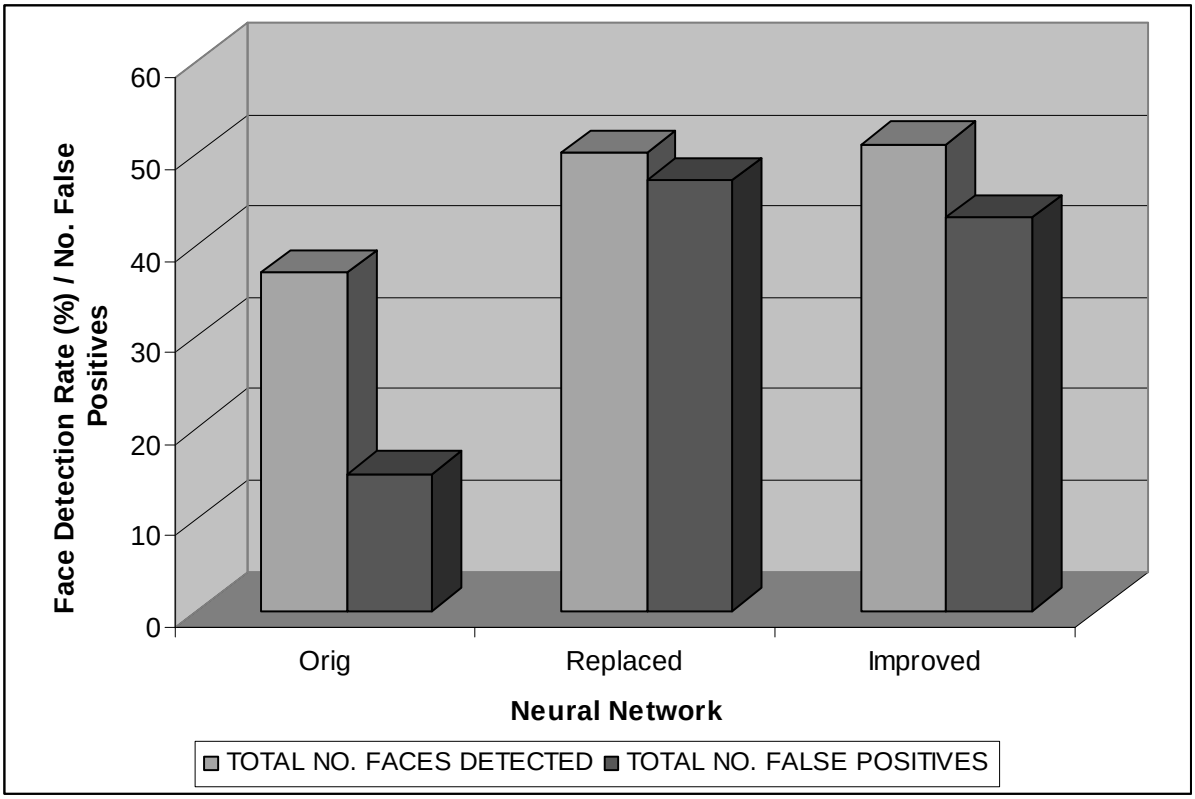


Figure 17b – A graph comparing the performance statistics (face detection rate and number of false positives), for the three networks.

6. Conclusion

Face detection has a many applications including Security applications,
Until recently, much of the work in the field of computer vision has focussed on face recognition, with very little research into face detection. Human face detection is often the first-step in the recognition process as detecting the location of a face in an image, prior to attempting recognition can focus computational resources on the face area of the image. Although a trivial task that human perform effortlessly, the task of face detection is a complex problem in computer vision, due the great multitude of variation present across faces. Several techniques have been proposed to solve this problem (as discussed in the Literature Survey, section 2), including what Hjelmås and Low [1] term Feature-Based approaches, and the more recent Image-Based approaches. Both categories of approaches offer systems worthy of recognition with promising results. Feature-Based Approaches, often applied to real-time systems, are often reliant on a priori knowledge of the face, which is used explicitly to detect features. The more robust Image-Based approaches are too computationally expensive for real-time systems, although systems using a hybrid of both approaches are being developed with encouraging results.

Arguably the stronger of the two approaches, image-based systems are being increasingly investigated, with much of the work involved in overcoming the limitations of computation time. Sanner's detector [41], evaluated throughout this project, is an example of an image-based approach, based on the Rowley et al [22] detector, which has helped set the standard to which others must comply. Sanner's detector implements some other key features of the Rowley et at detector [22], draws from the work of Sung and Poggio [19], and has been shown to produce encouraging results, despite the limitations stated.

Sanner included parameters to allow the detector to be fine tuned. The important trade-off between high detection rate and number of false positives, can be modified by adjusting the various parameters, and thus the system can be tweaked to suit a wide range of applications. This 'parameter tweaking' can be used to improve performance on individual images but in terms of general performance, the parameter values suggested by Sanner [41] proved optimal in most cases. If more was known about the application domain, parameters could be adjusted to improve the performance. For example, if the camera was in a fixed location, the pyramid

properties could be modified to prevent detections below or above a certain size (as the range of distances a face could be from the camera, restricts the size of a true face). This would minimise the number of false positives, and improve the computation time.

The changes made to the detector improved the detection rate, at the expense of a greater number of false positives. It would depend on the application, as to whether this was considered an actual improvement in performance or not, as some applications would sacrifice the detection of a few faces to keep the number of false positives as low as possible, whereas others would prefer to maximise the number of face detection regardless (e.g. a security system that identifies possible faces to be cross-checked with a database of existing offenders would rather 'waste time' processing false positives, than miss checking a true detection). For many systems though this somewhat primitive implementation would be unsuitable and would require extensive improvement (*see section 7*)

In conclusion, the system developed by Sanner [41] is a good example of a neural network based system, indicative of some of the more complex detectors in the field. It mirrors the strengths of the technology providing impressive classification results from a relatively small image training set, and also reflects the major limitations, mainly computational expense, and reliance on the training data. It illustrates well some of the key problems that developers of intelligent artificial face detection systems are faced with, not only in the field of neural networks but across the board.

7. Further Work

7.1 Further Improvements:

The main goal of this project was to introduce the field of face detection and implement a face detection system. An overview of the various approaches to the problem was compiled, and a description of Sanner's detector [41], along with a brief analysis was completed. The parameter values suggested by Sanner [41] were investigated although more extensive testing of the various parameters with a larger set of images is required. From the limitations identified by Sanner [41], improvements to the training data were made, and the performance of the modified system was compared to the original system's performance. The performance statistics show an improvement in the detection rate, but an accompanying increase in the number of false positives. The goal of any future improvement should be to improve the detection rate, minimise the number of false positives, and improve the speed of the detection process. A good place to start would be to aim to minimise the number of false positives. Improving the set of non-face examples would be extremely beneficial, as this is a major weakness in the current system. The compilation of a representative and adequate set of non-face examples is extremely difficult due to the immense variation present throughout non-face objects and scenery. The implementation of a bootstrapping technique would be ideal. Starting with no non-face data, a set of images is classified by the detector, and areas incorrectly identified as faces ('false positives'), are extracted and used as non-face examples. The system is re-trained, and the process repeated until the number of false detections falls below a threshold. This technique is though to construct a representative set of non-face examples, and would certainly offer performance improvements for this system. Finally it is recommended that the facescan function is modified slightly such that a counter is incremented each time a bounding box is drawn. This would allow more accurate results to be gathered, following a limitation identified during the manual inspection of images processed from the test set.

7.2 Project Extensions:

Although face detection is a key area of interest in the field of artificial machine intelligence, the usability of face recognition techniques lends the technology to a wider range of applications. Face detection techniques lie at the root of many face detection systems, identifying the location of faces, prior to attempting to recognise that face from a database of faces. Therefore a natural extension of this project, would be to extend the techniques identified to the field of face recognition.

8. References

1. Hjelmås, E. and Low, B.(2001). Face Detection: A Survey. Computer Vision and Image Understanding. 83, 236-274.

2. Sakai, T., Nagao, M. and Kanade, T. (1972). Computer analysis and classification of photographs of human face. Proc. First USA – Japan Computer Conference, 2.7.

3. Craw, I., Ellis, H. and Lishman J.R.(1987). Automatic extraction of face-feature. Pattern Recog. Lett. 183-187.

4. Govindaraju, V.(1996). Locating human faces in photographs. Int. J. Comput. Vision. 19.

5. Hoogenboom, R. and Lew, M.(1996). Face detection using local maxima. IEEE Proc. of 2nd Int. Conf. on Automatic face and gesture recognition, 334-339.

6. Yang, G. and Huang, T.S.(1994). Human face detection in a complex background. Pattern Recognition, 27, 53-63.

7. Terrillion, J.C., Shirazi, M., Fukamachi, H., and Akamatsu, S. (2000). Comparative performance of different skin chrominance models and chrominance spaces for the automatic detection of human faces in colour images. Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition.

8. De Silva, L.C., Aizawa, K., and Hatori, M.(1995). Detection and tracking of facial features by using a facial feature model and deformable circular template. IEICE Trans. Inform Systems. E78-D(9), 1995-1207.

9. Jeng, S.H.,Liao, H.Y.M., Han, C.C., Chern, M.Y., and Liu, Y.T.(1998). Facial feature detection using geometrical face model: an efficient approach. Pattern Recognition, 31.

10. Burl, M.C., Leung, T.K., and Perona, P.(1995). Face localization via shape statistics. Int. Workshop on Automatic Face and Gesture Recognition, Zurick.

11. Yow, K.C. and Cipolla, R.(1997). Feature-based human face detection. Image Vision Comput., 15(9).

12. Huang, C.L., and Chen, C.W.(1992). Human facial feature extraction for face interpretation and recognition. Pattern Recog, 25, 1435-1444.

13. Lam, K.M. and Yan, H.(1994). Fast greedy algorithm for locating head boundaries. Electron. Lett., 30, 21-22.

14. Gunn, S.R. and Nixon, M.S.(1994). A dual active contour for head and boundary extraction. IEE Colloquium on Image Processing for Biometric Measurement, 6/1.

15. Yuille, A.L., Hallinan, P.W., and Cohen, D.S.(1992). Feature extraction from faces using deformable templates. Int. J. Comput. Vision, 8, 99-111.

16. Lantis, A., Taylor, C.J., and Cootes, T.F.(1994). Automatic tracking, coding, and reconstruction of human faces, using flexible appearance models. IEEE Electron. Lett., 30, 1578-1579.

17. Sirovich, L. and Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. J. Opt. Soc. Amer., 4, 519-524.

18. Moghaddam, B. and Pentland, A. (1994). Face recognition using view-based and modular eigenspaces. Automatic Systems for the Identification of Humans, SPIE, 2277.

19. Sung, K.K. and Poggio, T. (1998). Example-based learning for view-based human face detection. IEEE Trans. Pattern Anal. Mach. Intelligence, 20, 39-51.

20. Yang, M.H., Ahuja, N., and Kriegman, D.(2000). Face detection using mixtures of linear subspacings. Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition.

21. Kohonen, T.(1995). Self-Organising Maps. Springer-Verlog, Berlin.

22. Rowley, H.A., Baluja, S., and Kanade, T.(1998). Neural network-based face detection. IEEE Trans. Pattern Anal. Mach. Intelligence, 20, 23-38.

23. Feraud, R., Bernier, O., and Collobert, D.(1997). A constrained generative model applied to face detection. Neural Process. Lett., 5, 73-81.

24. Lin, S.H., Kung, S.Y., Lin, L.J.(1997). Face recognition/detection by probabilistic decision-based neural network. IEEE Trans. Neural Networks, 8, 114-132.

25. Colmenarez, A.J. and Huang, T.S.(1997). Face detection with information-based maximum discrimination. IEEE Proc. of Int. Conf. on Comnputer Vision and Pattern Recognition, 6.

26. Osuna, E., Freund, R., and Girosi, F. (1997). Training support vector machines: An application to face detection. IEEE Proc. of Int. Conf. on Computers Vision and Pattern Recog, 33, 1369-1382.

27. Terrillion, J.C., Shirazi, M., Sadek, M., Fukamachi, H. and Akamatsu, S. (2000). Invariant face detection with support vector machines. Proceedings of the 15th International Conference on Pattern Recognition, 4, 4B.

28. Schneiderman, H. and Kanade, T. (1998). Probabilistic modelling of local appearance and spatial relationships for object recognition. IEEE Conference on Computer Vision and Pattern Recognition, 6.

29. Schneiderman, H. and Kanade, T. (2000). A statistical model for 3D object detection applied to faces and cars. IEEE Conference on Computer Vision and Pattern Recognition.

30. Huang, J., Gutta, S., and Wechsler, H.(1996). Detection of Human Faces Using Decision Trees. 2nd International Conf. of Automated Face and Hand Gesture Recognition.

31. Quinlan, J.R.(1986). The effect of noise on concept learning. Machine Learning: an Artificial Intelligence Approach, 2, 149-166.

32. Kirchberg, K.J., Jesorsky, O., and Frishholz, R.W.(2002). Genetic Model Optimization for Hausdorff Distance-Based Face Localization. Int. ECCV 2002 Workshop on Biometric Authentication, LNCS-2359, 103-111.

33. Elad, M., Hel-Or, Y., and Keshet, R.(2002). Rejection based classifier for face detection. Patern Recog. Letts, 23, 1459-1471.

34. Frischolz, R.W., and Dieckmann, U. (2000). A multimodal biometric identification system, IEE Comput. 33, 2.

35. Poulton, G.E., Oakes, N.A., Geers, D.G., and Qiao, R.Y.(1999). The CSIRO PC-Check system. Proceeding Second International Conference on Audio- and Video-based Biometric Person Authentication (AVBPA).

36. Turk, M., and Pentland, A. (1991). Eigenfaces for recognition. J. Cog. Neurosci. 3, 71-86.

37. Wang, C., Griebel, S.M., and Brandstein, M.S. (2000). Robust automatic video-conferencing with multiple cameras and microphones, Proc. IEEE International Conference on Multimedia and Expo.

38. Satoh, H., Nakamura, Y., and Kanade, T. (1999). Name-It: Naming and detecting faces in news video. IEEE Multimedia. 6, 22-35.

39. Wactlar, H.D., Kanade, T., Smith, M.A., and Stevens, S.M. (1996). Intelligent access to digital video: Informedia Project. IEEE Comput. 29(5), 46-52.

40. Frankel, C., Swain, M.J., and Athitsos, V.(1996). Webseer: An Image Search Engine for the World Wide Web. Technical Report TR 96-14, Computer Science Department, Univ. of Chicago.

41. Sanner, S. “Rowley-Baluja-Kanade Face Detector”. Available online at http://www.cs.toronto.edu/~ssanner/Software/Vision/Project.html

42. Rowley, H.A., Baluja, S., and Kanade, T.(1998). CMU Image Dataset. Available online at http://www-2.cs.cmu.edu/~har/faces.html

43. Mottram (2001). Sluggish Software, Available Online at http://www.fuzzgun.btinternet.co.uk/ or http://sourceforge.net/projects/rodney/

44. Browne, A. (1997). Neural Network Analysis, Architectures and Applications, IOP Publishing Limited.

45. Haykin, S. (1999). Neural Networks, A Comprehensive Foundation (2nd Edition) Prentice-Hall International Limited.

46. Lau, C. (1992). Neural Networks: Theoretical Foundations and Analysis, New York: IEEE Press.

47. Pratt, W.K. (2001). Digital Image Processing (3rd Edition), John Wiley & Sons.

48. Sánchez-Sinencio, E. and Lau, C.(1992). Artificial Neural Networks: Paradigms, Applications and Hardware Implementations, New York: IEEE Press.

49. Schalkoff, R.J. (1997). Artificial Neural Networks, McGraw-Hill Companies, Inc.

50. Werbos, P.J. (1994). The Roots of Back propagation, John Wiley & Sons.

9. Appendices

9.1 Original Code – Scott Sanner

9.1.1 facetrain.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Main training implementation of the Rowley-Beluja-Kanade face detector
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                                    Setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Neural net constants
FACE_T =  0.9;
FACE_F = -0.9;

% Load the image oval mask
MASK = buildmask;
NI   = size(find(MASK),1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                                Image Loading
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load the face images, normalize, and set training vectors
FACES = loadimages('./scaled/s', {'01' '02' '04' '07' '11' '99'}, ...
        {'c' 'l' 'r' 'n' 'g'}, 'PNG', 1);
FACES = augmentlr(FACES);
[NORM_FACES, SHADING] = normalize(FACES, MASK);
% Expand image set here
FACEV = buildimvector(NORM_FACES, MASK);
FACER = buildresvector(NORM_FACES, FACE_T);

% Load the non-face images, normalize, and set training vectors
NFACES = loadimages('./scaled/n', ...
       {'01' '02' '03' '04' '05' '06' '07' '08' '09' '10' '11' '12' '13' '14' ...
        '15' '16' '17' '18' '19' '20' '21' '22' '23' '24' '25' '26' '27' '28' ...
          '29' '30' '31' '32' '33' '34' '35' '36' '37' '38' '39' '40'}, ...
             {'x'}, 'PNG', 1);
NFACES = augmentlr(NFACES);
NFACES = augmentud(NFACES);
[NORM_NFACES, NSHADING] = normalize(NFACES, MASK);
% Expand image set here
NFACEV = buildimvector(NORM_NFACES, MASK);
NFACER = buildresvector(NORM_NFACES, FACE_F);

% Test images
AmyBW  = double(imread('./scaled/AmyBW.JPG'));
GradBW = double(imread('./scaled/GradBW.JPG'));
HeadsBW = double(imread('./scaled/HeadsBW.JPG'));

% Display images
% showimages(NORM_FACES,  5, 10, 1, 50, 1);
% showimages(NORM_NFACES, 5, 5, 1, 25, 2);
% pause;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                              Neural Net Training
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Build a neural net and train it
NET = createnn(NI, 25, 1, FACE_F, FACE_T);
[NET,PERF, ERR] = trainnn(NET,[NFACEV FACEV], [NFACER FACER], .10, 500);
figure(1); plot(PERF(:,1),PERF(:,2),'b-',PERF(:,1),PERF(:,3),'r-');
figure(2); plot(ERR (:,1),ERR (:,2),'b-',ERR (:,1),ERR (:,3),'r-');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                              Performance Testing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Test performance - for now just test on original images; extend to
% an image pyramid and return upper left corner of all 27x18 face
% bounding boxes
NUM_FACES  = size(FACES,2);
NUM_NFACES = size(NFACES,2);

t0 = clock; ferr = 0;
for i=1:NUM_FACES,
  TEST = classifynn(NET, FACES{i}, MASK,1,1);
  fprintf(1, '(Target, Test, Match): (%f,%f,%d)\n', FACE_T, TEST, TEST > 0);
  ferr = ferr + (TEST < 0);
end

nferr = 0;
for i=1:NUM_NFACES,
  TEST = classifynn(NET, NFACES{i}, MASK,1,1);
  fprintf(1, '(Target, Test, Match): (%f,%f,%d)\n', FACE_F, TEST, TEST < 0);
  nferr = nferr + (TEST > 0);
end

fprintf(1, '\n(Face Err, Nonface Err, Total Err): (%1.3f,%1.3f,%1.3f)\n', ...
       ferr./NUM_FACES, nferr./NUM_NFACES, (ferr + nferr)./(NUM_FACES +
NUM_NFACES));
fprintf(1, 'Time to classify %d images: %5.3f\n', NUM_FACES+NUM_NFACES,
etime(clock,t0));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

9.1.2 facescan.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Build an image resolution pyramid and scan the pyramid for
%         faces given the neural net, image, mask, and threshold -1<THR<1
%
% [RECT, IMR] = facescan(NET, IM, MASK, THR, LEVELS, START, SCALEFACT, STEP)
function [RECT, IMR] = facescan(NET, IM, MASK, THR, LEVELS, START, SCALEFACT,
STEP)

% START can be 1, but use to ignore smaller rectangles (level to start at)
% THR is good around .4 - .6
% STEP is good at 2
% LEVELS is good at 6 (number of pyramid levels with 1 being initial image)
% SCALEFACT is good at 1.2

% Setup
PYR_MAX = LEVELS; % A good choice is 6
MROWS = size(MASK,1);
MCOLS = size(MASK,2);
IROWS = size(IM, 1);
ICOLS = size(IM, 2);
RECT = [];

% Build the image pyramid
SCALE = SCALEFACT; % A good choice is 1.2
PYR{1} = IM;
XRANGE{1} = 1:1:ICOLS;
YRANGE{1} = 1:1:IROWS;
[MX{1},MY{1}] = meshgrid(XRANGE{1}, YRANGE{1});
for i=2:PYR_MAX,
       XRANGE{i} = 1:SCALE.^(i-1):ICOLS;
       YRANGE{i} = 1:SCALE.^(i-1):IROWS;
       [MX{i},MY{i}] = meshgrid(XRANGE{i}, YRANGE{i});
       PYR{i} = interp2(MX{1}, MY{1}, PYR{1}, MX{i}, MY{i});
end

% View pyramid
%figure(1);
%colormap(gray);
%showimages(PYR, 2, 3, 1, 6, 1);
%drawnow;
%pause;

% Scan the pyramid
for im_num = START:PYR_MAX,
  fprintf(1, '\n\nImage: %d\n', im_num);
  for im_row = 1:STEP:size(PYR{im_num},1)-MROWS+1,
    fprintf(1, ' R:%d', im_row);
    for im_col = 1:STEP:size(PYR{im_num},2)-MCOLS+1,
       TEST = classifynn(NET, PYR{im_num}, MASK, im_row, im_col);
        if (TEST > THR)
          fprintf(1, '\n  -(INUM,R,C,TEST): [%d] (%d,%d) => %5.3f   ',im_num,
im_row, im_col, TEST);
          RECT = [RECT; (im_row/size(YRANGE{im_num},2))*size(YRANGE{1},2), ...
                        (im_col/size(XRANGE{im_num},2))*size(XRANGE{1},2), ...
                        ((im_row+MROWS-
1)/size(YRANGE{im_num},2))*size(YRANGE{1},2), ...
                        ((im_col+MCOLS-
1)/size(XRANGE{im_num},2))*size(XRANGE{1},2), ...
                        TEST];
       end
    end
  end
end
```

```
% Plot the bounding boxes in an image to return
IMR = IM;
for i=1:size(RECT,1),
  SR = ceil(RECT(i,1));
  ER = ceil(RECT(i,3));
  SC = ceil(RECT(i,2));
  EC = ceil(RECT(i,4));
  IMR(SR,SC:EC) = 0;
  IMR(ER,SC:EC) = 0;
  IMR(SR:ER,SC) = 0;
  IMR(SR:ER,EC) = 0;
end

% Plot the image
figure(2);
colormap(gray);
imagesc(IMR);
drawnow;
```

9.1.3 augmentlr.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Augments an image array with the L<->R reversed images
%
% IM_NEW = augmentlr(IM)
function IM = augmentlr(IM)

num = size(IM,2);
nrows = size(IM{1},1);
ncols = size(IM{1},2);

for i=1:num,
      IM{i+num} = IM{i}(1:1:nrows,ncols:-1:1);
end
```

9.1.4 augmentud.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Augments an image array with the upside down reversed images
%
% IM_NEW = augment(IM)
function IM = augmentud(IM)

num = size(IM,2);
nrows = size(IM{1},1);
ncols = size(IM{1},2);

for i=1:num,
      IM{i+num} = IM{i}(nrows:-1:1,1:1:ncols);
end
```

9.1.5 buildimvector.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Builds an image vector from an image array, uses the mask
%         to find the appropriate indices
%
% IMVECTOR = buildimvector(IM, MASK)
function IMVECTOR = buildimvector(IM, MASK)

pics = size(IM,2);
INDICES = find(MASK);

IMVECTOR = zeros(size(find(MASK),1),pics);
for i=1:pics,
  IMVECTOR(:,i) = IM{i}(INDICES);
end
```

9.1.6 buildmask.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Builds an oval mask for face images
%
% MASK = buildmask()
function MASK = buildmask()

% An 18x27 mask
MASK = ...
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; ...
      0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0; ...
      0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0; ...
      0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
      0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
      0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
      0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
      0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0; ...
      0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
      0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
      0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0; ...
      0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
      0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0; ...
      0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0; ...
      0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0; ...
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0      ];
```

9.1.7 buildresvector.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Builds a target result vector for an image array
%
% RESVECTOR = buildresvector(IM, val)
function RESVECTOR = buildresvector(IM, val)

RESVECTOR = ones(1,size(IM,2)).*val;
```

9.1.8 classifynn.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   From a given image, returns the classification value
%
% TEST = classifynn(NET, IM, MASK)
function TEST = classifynn(NET, IM, MASK, srow, scol)

% First normalize the face

V{1} = IM(srow:srow+26,scol:scol+17);
[V, PLANE] = normalize(V, MASK);

% Now test it
TEST = simnn(NET, V{1}(find(MASK)));
```

9.1.9 createnn.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Creates a neural net with the given parameters
%
% NET = createnn(input, hidden, output, min, max)
function NET = createnn(input, hidden, output, min, max)

PR = repmat([min,max],input,1);
S = [hidden output];
T = {'tansig' 'tansig'};
NET = newff(PR,S,T,'traingdm');
```

9.1.10 loadimages.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Loads a set of images into an array given
%         a prefix, suffix, and highest index filename.
%
% IM = loadimages(prefix, subject, letter, suffix, show)
function IM = loadimages(prefix, subject, letter, suffix, show)
clear IM;
% Load the image set
for i = 1:size(subject,2),
   for j = 1:size(letter,2),
      IM{(i-1)*size(letter,2)+j} = double(imread([prefix, subject{i}, '-',
letter{j}, '.', suffix]));
   end
end
```

9.1.11 normalize.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Normalizes an image by removing shading plane and adjusting
%         histogram to scale to min/max [0,1]
%
% [OUT] = normalize(IN, MASK)
function [OUT, SHADING] = normalize(IN, MASK)

% Retrieve the indices for the given mask
IND = find(MASK);
% Set up matrices for planar projection calculation
% i.e. Ax = B  so  x = (A'*A)^-1 * A'*B
x = 1:1:size(IN{1},2);
y = 1:1:size(IN{1},1);
[mx,my] = meshgrid(x,y);
mxc = mx(IND);
myc = my(IND);
mcc = ones(size(myc));
A = [mxc, myc, mcc];

% Cycle through each image removing shading plane
% and adjusting histogram
for i=1:size(IN,2),
   % Calculate plane: z = ax + by + c
   B = IN{i}(IND);
   x = inv(A'*A)*A'*B;
   a = x(1); b = x(2); c = x(3);
   %This is the color plane itself
   SHADING{i} = mx.*a + my.*b + c;
   %This is the image minus the color plane
   %(the constant will be normalized out in histogram recentering)
   OUT{i} = IN{i} - (mx.*a + my.*b + c);
   % Now, recenter the histogram
   maximum = max(max(OUT{i}.*MASK));
   minimum = min(min(OUT{i}.*MASK));   %minimum = min(min(OUT{i}))
   diff = maximum - minimum;
   if (diff == 0)
       fprintf('\n\nNo Change Required\n\n');
   else
       OUT{i} = ((OUT{i}-minimum)./diff).*MASK;
   end
end
```

9.1.12 showimages.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Displays an image array
%
% showimages(IM, xdim, ydim, imgstr, imgend, fign)
function showimages(IM, xdim, ydim, imgstr, imgend, fign)

% Show the image set if fign is valid
if (fign>0)
   figure(fign);
   for i=imgstr:imgend,
      subplot(xdim,ydim,i-imgstr+1);
      imagesc(IM{i});
      colormap gray;
   end
end
```

9.1.13 simnn.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Simulates a net on the image array, returning a vector of
%         the results... either -1 or 1
% RESULT = simnn(NET, IMVECTOR)
function RESULT = simnn(NET, IMVECTOR)

% OK, do I really need this function?
RESULT = sim(NET, IMVECTOR);
```

9.1.14 testnn.m

```
function testnn()

% Min/Max values for each input
PR = [0 1;
      0 1];
% Size of hidden/output layer
S = [5 1];
% Cols - Input vectors, each col diff. vector
TI = [0 0 1 1;
      0 1 0 1];
% Cols - Target output for each vector
TO = [-.9 .9 .9 -.9];
% Unit type for hidden and output layers
T = {'tansig' 'tansig'};
% Create neural net
net = newff(PR,S,T,'traingdm');

% Simulate and plot
Y1 = sim(net, TI);
subplot(1,3,1);
plot([1 2 3 4], Y1);

% Set number of training epochs and error
subplot(1,3,2);
net.trainParam.epochs = 500;
net.trainParam.goal   = 0.001;

% Train
[net, tr] = train(net, TI, TO);
%tr.perf  % - trace of performance error [epochs+1]

% Simulate and plot
Y1 = sim(net, TI);
subplot(1,3,3);
plot([1 2 3 4], Y1);
```

9.1.15 trainnn.m

```
% Author: Scott Sanner
% Email:  ssanner@cs.stanford.edu
% Course: CS223B, Winter
% Desc:   Trains a neural net given a training set with target values
%         and randomly selects training and validation data, training
%         either for the max number of networks, network convergence,
%         or increase in validation error.  Returns network and
%         performance.
%
% [NET,PERF, ERR] = trainnn(NET,IMVECTOR,TVECTOR,percent_val,iter)
function [NET,PERF, ERR] = trainnn(NET,IMVECTOR,TVECTOR,percent_val,iter)

% Setup validation and test sets
N_IMS = size(IMVECTOR,2);
CHOICE  = rand(1,N_IMS) > percent_val;
V_TRAIN = find(CHOICE);
V_VALID = find(1-CHOICE);
IM_TRAIN_V = IMVECTOR(:,V_TRAIN);
IM_TRAIN_R = TVECTOR (:,V_TRAIN);
IM_VALID_V = IMVECTOR(:,V_VALID);
IM_VALID_R = TVECTOR (:,V_VALID);
fprintf(1,'Training set quantity:   #%d\n', size(IM_TRAIN_V,2));
fprintf(1,'Validation set quantity: #%d\n', size(IM_VALID_V,2));

% Setup net parameters
NET.trainParam.epochs = 1;
NET.trainParam.goal   = 0.0001;
TRAIN_OUT = simnn(NET,IM_TRAIN_V);
VALID_OUT = simnn(NET,IM_VALID_V);
PERF = [];
ERR  = [];
PERF = [PERF; 0, (1-(sum(abs(TRAIN_OUT-IM_TRAIN_R)./1.8)./(size(TRAIN_OUT,2)))),
...
                (1-(sum(abs(VALID_OUT-IM_VALID_R)./1.8)./(size(VALID_OUT,2))))];
ERR  = [ERR;  0, ((sum(abs(TRAIN_OUT-IM_TRAIN_R))./(size(TRAIN_OUT,2))).^2), ...
                ((sum(abs(VALID_OUT-IM_VALID_R))./(size(VALID_OUT,2))).^2)];

% Train for specified number of iterations
for i=1:iter,
  fprintf(1,'Starting iteration %d\n', i);
  drawnow;
  [NET, TR] = train(NET, IM_TRAIN_V, IM_TRAIN_R);
  TRAIN_OUT = simnn(NET,IM_TRAIN_V);
  VALID_OUT = simnn(NET,IM_VALID_V);
  %[TRAIN_OUT; IM_TRAIN_R]
  %[VALID_OUT; IM_VALID_R]
  PERF = [PERF; i, (1-(sum(abs(TRAIN_OUT-IM_TRAIN_R)./1.8)./(size(TRAIN_OUT,2)))),
...
                  (1-(sum(abs(VALID_OUT-
IM_VALID_R)./1.8)./(size(VALID_OUT,2))))];
  ERR  = [ERR;  i, ((sum(abs(TRAIN_OUT-IM_TRAIN_R))./(size(TRAIN_OUT,2))).^2), ...
                  ((sum(abs(VALID_OUT-IM_VALID_R))./(size(VALID_OUT,2))).^2)];
end
```

9.2 Other Code

9.2.1 massclassify.m

```
% Author: Phil Brimblecombe
% Email:  ee01pb@surrey.ac.uk
% Course: MEng Electronic Engineering
% Desc:   Classifies the existing training data and reports performance statistics

FaceYes = 0;
FaceNo = 0;
NFaceYes = 0;
NFaceNo = 0;
fprintf ('\n Face Examples \n \n');
for (i =1:NUM_FACES),
RESULT_FACE{i} = classifynn(NET, FACES{i}, MASK, 1,1)
if (RESULT_FACE{i} > 0)
       FaceYes = (FaceYes +1);
   end
 if  (RESULT_FACE{i} < 0)
       FaceNo = (FaceNo + 1);
   end
end
fprintf ('\n Non-Face Examples \n \n');
for (i =1:NUM_NFACES),
RESULT_NFACE{i} = classifynn(NET, NFACES{i}, MASK, 1,1)
if (RESULT_NFACE{i} < 0)
    NFaceYes = (NFaceYes + 1);
 end
if  (RESULT_NFACE{i} > 0)
    NFaceNo = (NFaceNo + 1);
 end
end
fprintf ('RESULTS:\n\n');
fprintf('Of the %d face examples: \n %d were correctly classified \n %d were
misclassified \n\n', NUM_FACES, FaceYes, FaceNo);
fprintf('Of the %d non-face examples: \n %d were correctly classified \n %d were
misclassified \n\n', NUM_NFACES, NFaceYes, NFaceNo);
fprintf ('SUMMARY:\n\n');
fprintf('Correct Face Detection Rate of %3.1f (percent)\n',
(FaceYes/NUM_FACES)*100 );
fprintf('Correct Non-Face Detection Rate of %3.1f (percent)\n',
(NFaceYes/NUM_NFACES)*100);
fprintf('Total Detection Rate of %3.1f (percent)\n',
((FaceYes+NFaceYes)/(NUM_FACES+NUM_NFACES))*100);
```

9.2.2 showimages.m

```
% Author: Phil Brimblecombe
% Email:  ee01pb@surrey.ac.uk
% Course: MEng Electronic Engineering
% Desc:   Displays single images once classified.

function NOTHING = showimage(PARAM)

figure(1);
colormap (gray);
imagesc(PARAM);
drawnow;
```

9.2.3 pyramidtest.m

```
% Author: Phil Brimblecombe
% Email:  ee01pb@surrey.ac.uk
% Course: MEng Electronic Engineering
% Desc:   Automated testing of pyramid parameters

%LOAD TEST IMAGES
FF3 = double (imread ('FF3.jpg'));
FF8 = double (imread ('FF8.jpg'));
NF5 = double (imread ('NF5.gif'));
NF3 = double (imread ('NF3.gif'));

% Vary Levels of Pyramid between 1 and 20 at 2 layer increments
[RECTFF3_L1, IMRFF3_L1] = facescan (NET, FF3, MASK, 0.7, 1, 1, 1.2, 2)
[RECTFF8_L1, IMRFF8_L1] = facescan (NET, FF8, MASK, 0.7, 1, 1, 1.2, 2)
[RECTNF5_L1, IMRNF5_L1] = facescan (NET, NF5, MASK, 0.7, 1, 1, 1.2, 2)
[RECTNF3_L1, IMRNF3_L1] = facescan (NET, NF3, MASK, 0.7, 1, 1, 1.2, 2)

[RECTFF3_L2, IMRFF3_L2] = facescan (NET, FF3, MASK, 0.7, 2, 1, 1.2, 2)
[RECTFF8_L2, IMRFF8_L2] = facescan (NET, FF8, MASK, 0.7, 2, 1, 1.2, 2)
[RECTNF5_L2, IMRNF5_L2] = facescan (NET, NF5, MASK, 0.7, 2, 1, 1.2, 2)
[RECTNF3_L2, IMRNF3_L2] = facescan (NET, NF3, MASK, 0.7, 2, 1, 1.2, 2)

[RECTFF3_L4, IMRFF3_L4] = facescan (NET, FF3, MASK, 0.7, 4, 1, 1.2, 2)
[RECTFF8_L4, IMRFF8_L4] = facescan (NET, FF8, MASK, 0.7, 4, 1, 1.2, 2)
[RECTNF5_L4, IMRNF5_L4] = facescan (NET, NF5, MASK, 0.7, 4, 1, 1.2, 2)
[RECTNF3_L4, IMRNF3_L4] = facescan (NET, NF3, MASK, 0.7, 4, 1, 1.2, 2)

[RECTFF3_L6, IMRFF3_L6] = facescan (NET, FF3, MASK, 0.7, 6, 1, 1.2, 2)
[RECTFF8_L6, IMRFF8_L6] = facescan (NET, FF8, MASK, 0.7, 6, 1, 1.2, 2)
[RECTNF5_L6, IMRNF5_L6] = facescan (NET, NF5, MASK, 0.7, 6, 1, 1.2, 2)
[RECTNF3_L6, IMRNF3_L6] = facescan (NET, NF3, MASK, 0.7, 6, 1, 1.2, 2)

[RECTFF3_L8, IMRFF3_L8] = facescan (NET, FF3, MASK, 0.7, 8, 1, 1.2, 2)
[RECTFF8_L8, IMRFF8_L8] = facescan (NET, FF8, MASK, 0.7, 8, 1, 1.2, 2)
[RECTNF5_L8, IMRNF5_L8] = facescan (NET, NF5, MASK, 0.7, 8, 1, 1.2, 2)
[RECTNF3_L8, IMRNF3_L8] = facescan (NET, NF3, MASK, 0.7, 8, 1, 1.2, 2)

[RECTFF3_L10, IMRFF3_L10] = facescan (NET, FF3, MASK, 0.7, 10, 1, 1.2, 2)
[RECTFF8_L10, IMRFF8_L10] = facescan (NET, FF8, MASK, 0.7, 10, 1, 1.2, 2)
[RECTNF5_L10, IMRNF5_L10] = facescan (NET, NF5, MASK, 0.7, 10, 1, 1.2, 2)
[RECTNF3_L10, IMRNF3_L10] = facescan (NET, NF3, MASK, 0.7, 10, 1, 1.2, 2)

[RECTFF3_L12, IMRFF3_L12] = facescan (NET, FF3, MASK, 0.7, 12, 1, 1.2, 2)
[RECTFF8_L12, IMRFF8_L12] = facescan (NET, FF8, MASK, 0.7, 12, 1, 1.2, 2)
[RECTNF5_L12, IMRNF5_L12] = facescan (NET, NF5, MASK, 0.7, 12, 1, 1.2, 2)
[RECTNF3_L12, IMRNF3_L12] = facescan (NET, NF3, MASK, 0.7, 12, 1, 1.2, 2)

[RECTFF3_L14, IMRFF3_L14] = facescan (NET, FF3, MASK, 0.7, 14, 1, 1.2, 2)
[RECTFF8_L14, IMRFF8_L14] = facescan (NET, FF8, MASK, 0.7, 14, 1, 1.2, 2)
[RECTNF5_L14, IMRNF5_L14] = facescan (NET, NF5, MASK, 0.7, 14, 1, 1.2, 2)
[RECTNF3_L14, IMRNF3_L14] = facescan (NET, NF3, MASK, 0.7, 14, 1, 1.2, 2)

[RECTFF3_L16, IMRFF3_L16] = facescan (NET, FF3, MASK, 0.7, 16, 1, 1.2, 2)
[RECTFF8_L16, IMRFF8_L16] = facescan (NET, FF8, MASK, 0.7, 16, 1, 1.2, 2)
[RECTNF5_L16, IMRNF5_L16] = facescan (NET, NF5, MASK, 0.7, 16, 1, 1.2, 2)
[RECTNF3_L16, IMRNF3_L16] = facescan (NET, NF3, MASK, 0.7, 16, 1, 1.2, 2)

[RECTFF3_L18, IMRFF3_L18] = facescan (NET, FF3, MASK, 0.7, 18, 1, 1.2, 2)
[RECTFF8_L18, IMRFF8_L18] = facescan (NET, FF8, MASK, 0.7, 18, 1, 1.2, 2)
[RECTNF5_L18, IMRNF5_L18] = facescan (NET, NF5, MASK, 0.7, 18, 1, 1.2, 2)
[RECTNF3_L18, IMRNF3_L18] = facescan (NET, NF3, MASK, 0.7, 18, 1, 1.2, 2)

[RECTFF3_L20, IMRFF3_L20] = facescan (NET, FF3, MASK, 0.7, 20, 1, 1.2, 2)
[RECTFF8_L20, IMRFF8_L20] = facescan (NET, FF8, MASK, 0.7, 20, 1, 1.2, 2)
[RECTNF5_L20, IMRNF5_L20] = facescan (NET, NF5, MASK, 0.7, 20, 1, 1.2, 2)
[RECTNF3_L20, IMRNF3_L20] = facescan (NET, NF3, MASK, 0.7, 20, 1, 1.2, 2)
```

```
[RECTNF5_SF11, IMRNF5_SF11] = facescan (NET, NF5, MASK, 0.7, 10, 1, 1.1, 2)
[RECTNF3_SF11, IMRNF3_SF11] = facescan (NET, NF3, MASK, 0.7, 10, 1, 1.1, 2)

[RECTFF3_SF12, IMRFF3_SF12] = facescan (NET, FF3, MASK, 0.7, 10, 1, 1.2, 2)
[RECTFF8_SF12, IMRFF8_SF12] = facescan (NET, FF8, MASK, 0.7, 10, 1, 1.2, 2)
[RECTNF5_SF12, IMRNF5_SF12] = facescan (NET, NF5, MASK, 0.7, 10, 1, 1.2, 2)
[RECTNF3_SF12, IMRNF3_SF12] = facescan (NET, NF3, MASK, 0.7, 10, 1, 1.2, 2)

[RECTFF3_SF13, IMRFF3_SF13] = facescan (NET, FF3, MASK, 0.7, 10, 1, 1.3, 2)
[RECTFF8_SF13, IMRFF8_SF13] = facescan (NET, FF8, MASK, 0.7, 10, 1, 1.3, 2)
[RECTNF5_SF13, IMRNF5_SF13] = facescan (NET, NF5, MASK, 0.7, 10, 1, 1.3, 2)
[RECTNF3_SF13, IMRNF3_SF13] = facescan (NET, NF3, MASK, 0.7, 10, 1, 1.3, 2)

[RECTFF3_SF14, IMRFF3_SF14] = facescan (NET, FF3, MASK, 0.7, 10, 1, 1.4, 2)
[RECTFF8_SF14, IMRFF8_SF14] = facescan (NET, FF8, MASK, 0.7, 10, 1, 1.4, 2)
[RECTNF5_SF14, IMRNF5_SF14] = facescan (NET, NF5, MASK, 0.7, 10, 1, 1.4, 2)
[RECTNF3_SF14, IMRNF3_SF14] = facescan (NET, NF3, MASK, 0.7, 10, 1, 1.4, 2)

[RECTFF3_SF15, IMRFF3_SF15] = facescan (NET, FF3, MASK, 0.7, 10, 1, 1.5, 2)
[RECTFF8_SF15, IMRFF8_SF15] = facescan (NET, FF8, MASK, 0.7, 10, 1, 1.5, 2)
[RECTNF5_SF15, IMRNF5_SF15] = facescan (NET, NF5, MASK, 0.7, 10, 1, 1.5, 2)
[RECTNF3_SF15, IMRNF3_SF15] = facescan (NET, NF3, MASK, 0.7, 10, 1, 1.5, 2)

%Varying the START parameter (to ignore smaller rectangles)
[RECTFF3_ST1, IMRFF3_ST1] = facescan (NET, FF3, MASK, 0.7, 10, 1, 1.2, 2)
[RECTFF8_ST1, IMRFF8_ST1] = facescan (NET, FF8, MASK, 0.7, 10, 1, 1.2, 2)
[RECTNF5_ST1, IMRNF5_ST1] = facescan (NET, NF5, MASK, 0.7, 10, 1, 1.2, 2)
[RECTNF3_ST1, IMRNF3_ST1] = facescan (NET, NF3, MASK, 0.7, 10, 1, 1.2, 2)

[RECTFF3_ST2, IMRFF3_ST2] = facescan (NET, FF3, MASK, 0.7, 10, 2, 1.2, 2)
[RECTFF8_ST2, IMRFF8_ST2] = facescan (NET, FF8, MASK, 0.7, 10, 2, 1.2, 2)
[RECTNF5_ST2, IMRNF5_ST2] = facescan (NET, NF5, MASK, 0.7, 10, 2, 1.2, 2)
[RECTNF3_ST2, IMRNF3_ST2] = facescan (NET, NF3, MASK, 0.7, 10, 2, 1.2, 2)

[RECTFF3_ST3, IMRFF3_ST3] = facescan (NET, FF3, MASK, 0.7, 10, 3, 1.2, 2)
[RECTFF8_ST3, IMRFF8_ST3] = facescan (NET, FF8, MASK, 0.7, 10, 3, 1.2, 2)
[RECTNF5_ST3, IMRNF5_ST3] = facescan (NET, NF5, MASK, 0.7, 10, 3, 1.2, 2)
[RECTNF3_ST3, IMRNF3_ST3] = facescan (NET, NF3, MASK, 0.7, 10, 3, 1.2, 2)

[RECTFF3_ST4, IMRFF3_ST4] = facescan (NET, FF3, MASK, 0.7, 10, 4, 1.2, 2)
[RECTFF8_ST4, IMRFF8_ST4] = facescan (NET, FF8, MASK, 0.7, 10, 4, 1.2, 2)
[RECTNF5_ST4, IMRNF5_ST4] = facescan (NET, NF5, MASK, 0.7, 10, 4, 1.2, 2)
[RECTNF3_ST4, IMRNF3_ST4] = facescan (NET, NF3, MASK, 0.7, 10, 4, 1.2, 2)

[RECTFF3_ST5, IMRFF3_ST5] = facescan (NET, FF3, MASK, 0.7, 10, 5, 1.2, 2)
[RECTFF8_ST5, IMRFF8_ST5] = facescan (NET, FF8, MASK, 0.7, 10, 5, 1.2, 2)
[RECTNF5_ST5, IMRNF5_ST5] = facescan (NET, NF5, MASK, 0.7, 10, 5, 1.2, 2)
[RECTNF3_ST5, IMRNF3_ST5] = facescan (NET, NF3, MASK, 0.7, 10, 5, 1.2, 2)

[RECTFF3_ST6, IMRFF3_ST6] = facescan (NET, FF3, MASK, 0.7, 10, 6, 1.2, 2)
[RECTFF8_ST6, IMRFF8_ST6] = facescan (NET, FF8, MASK, 0.7, 10, 6, 1.2, 2)
[RECTNF5_ST6, IMRNF5_ST6] = facescan (NET, NF5, MASK, 0.7, 10, 6, 1.2, 2)
[RECTNF3_ST6, IMRNF3_ST6] = facescan (NET, NF3, MASK, 0.7, 10, 6, 1.2, 2)

[RECTFF3_ST7, IMRFF3_ST7] = facescan (NET, FF3, MASK, 0.7, 10, 7, 1.2, 2)
[RECTFF8_ST7, IMRFF8_ST7] = facescan (NET, FF8, MASK, 0.7, 10, 7, 1.2, 2)
[RECTNF5_ST7, IMRNF5_ST7] = facescan (NET, NF5, MASK, 0.7, 10, 7, 1.2, 2)
[RECTNF3_ST7, IMRNF3_ST7] = facescan (NET, NF3, MASK, 0.7, 10, 7, 1.2, 2)

[RECTFF3_ST8, IMRFF3_ST8] = facescan (NET, FF3, MASK, 0.7, 10, 8, 1.2, 2)
[RECTFF8_ST8, IMRFF8_ST8] = facescan (NET, FF8, MASK, 0.7, 10, 8, 1.2, 2)
[RECTNF5_ST8, IMRNF5_ST8] = facescan (NET, NF5, MASK, 0.7, 10, 8, 1.2, 2)
[RECTNF3_ST8, IMRNF3_ST8] = facescan (NET, NF3, MASK, 0.7, 10, 8, 1.2, 2)

[RECTFF3_ST9, IMRFF3_ST9] = facescan (NET, FF3, MASK, 0.7, 10, 9, 1.2, 2)
[RECTFF8_ST9, IMRFF8_ST9] = facescan (NET, FF8, MASK, 0.7, 10, 9, 1.2, 2)
[RECTNF5_ST9, IMRNF5_ST9] = facescan (NET, NF5, MASK, 0.7, 10, 9, 1.2, 2)
[RECTNF3_ST9, IMRNF3_ST9] = facescan (NET, NF3, MASK, 0.7, 10, 9, 1.2, 2)
```

9.2.4 nonfacecrop.m

```
% Author: Scott Sanner (Modified by Phil Brimblecombe 2004)
% Email:  ssanner@cs.stanford.edu / ee01pb@surrey.ac.uk
% Course: CS223B, Winter / MEng Electronic Engineering
% Desc:   To be used with non-face or scenery images only.  Is an extension
%         of "facescan" using same parameters.  Searches image for "faces".  Those
%         identified are false positives.  Provides cropping facility to allow
%         these false positives to be crudely extracted.

function [RECT, IMR] = nonfacecrop(NET, IM, MASK, THR, LEVELS, START, SCALEFACT,
STEP)

% Setup
PYR_MAX = LEVELS; % A good choice is 6
MROWS = size(MASK,1);
MCOLS = size(MASK,2);
IROWS = size(IM, 1);
ICOLS = size(IM, 2);
RECT = [];

% Build the image pyramid
SCALE = SCALEFACT; % A good choice is 1.2
PYR{1} = IM;
XRANGE{1} = 1:1:ICOLS;
YRANGE{1} = 1:1:IROWS;
[MX{1},MY{1}] = meshgrid(XRANGE{1}, YRANGE{1});
for i=2:PYR_MAX,
      XRANGE{i} = 1:SCALE.^(i-1):ICOLS;
      YRANGE{i} = 1:SCALE.^(i-1):IROWS;
      [MX{i},MY{i}] = meshgrid(XRANGE{i}, YRANGE{i});
      PYR{i} = interp2(MX{1}, MY{1}, PYR{1}, MX{i}, MY{i});
end

% View pyramid
%figure(1);
%colormap(gray);
%showimages(PYR, 2, 3, 1, 6, 1);
%drawnow;
%pause;

% Scan the pyramid
for im_num = START:PYR_MAX,
  fprintf(1, '\n\nImage: %d\n', im_num);
  for im_row = 1:STEP:size(PYR{im_num},1)-MROWS+1,
    fprintf(1, ' R:%d', im_row);
    for im_col = 1:STEP:size(PYR{im_num},2)-MCOLS+1,
     fname = (((im_num)*100) + ((im_row)*10) + (im_col));
      TEST = classifynn(NET, PYR{im_num}, MASK, im_row, im_col);
      if (TEST > THR)
       fprintf(1, '\n  -(INUM,R,C,TEST): [%d] (%d,%d) => %5.3f   ',im_num, im_row,
im_col, TEST);
         RECT = [RECT; (im_row/size(YRANGE{im_num},2))*size(YRANGE{1},2), ...
                       (im_col/size(XRANGE{im_num},2))*size(XRANGE{1},2), ...
                       ((im_row+MROWS-
1)/size(YRANGE{im_num},2))*size(YRANGE{1},2), ...
                       ((im_col+MCOLS-
1)/size(XRANGE{im_num},2))*size(XRANGE{1},2), ...
                       TEST];
      end
    end
  end
end
```

```
% Plot the bounding boxes in an image to return
IMR = IM;
for i=1:size(RECT,1),
  SR = ceil(RECT(i,1));
  ER = ceil(RECT(i,3));
  SC = ceil(RECT(i,2));
  EC = ceil(RECT(i,4));
  IMR(SR,SC:EC) = 0; %Horiz Line
  IMR(ER,SC:EC) = 0; %Horiz Line
  IMR(SR:ER,SC) = 0; %Vert Line
  IMR(SR:ER,EC) = 0;
end

% Plot the image
imwrite(IMR, 'facescan_output.png');
figure(2);
colormap(gray);
imagesc(IMR);
drawnow;

for i=1:size(RECT,1),
figure(2);
colormap(gray);
imagesc(IMR);
drawnow;

% Crop the image
CROP{i} = imcrop (IMR);
if (i==1) FN = 'Cropped01.PNG'; end;
if (i==2) FN = 'Cropped02.PNG'; end;
if (i==3) FN = 'Cropped03.PNG'; end;
if (i==4) FN = 'Cropped04.PNG'; end;
if (i==5) FN = 'Cropped05.PNG'; end;
if (i==6) FN = 'Cropped06.PNG'; end;
if (i==7) FN = 'Cropped07.PNG'; end;
if (i==8) FN = 'Cropped08.PNG'; end;
if (i==9) FN = 'Cropped09.PNG'; end;
if (i==10) FN = 'Cropped10.PNG'; end;
imwrite (CROP{i}, (FN));
end
```

9.2.5 testset01.m

```
% Author: Phil Brimblecombe
% Email:  ee01pb@surrey.ac.uk
% Course: MEng Electronic Engineering
% Desc:   This script was written to test the performance of networks with
%         TestSet01, images from CMU-VASC - Statistics evaluated manually

%LOAD TEST IMAGES
TS01_01 = double (imread ('./testset01/TS01-01.pgm'));
TS01_02 = double (imread ('./testset01/TS01-02.pgm'));
TS01_03 = double (imread ('./testset01/TS01-03.pgm'));
TS01_04 = double (imread ('./testset01/TS01-04.pgm'));
TS01_05 = double (imread ('./testset01/TS01-05.pgm'));
TS01_06 = double (imread ('./testset01/TS01-06.pgm'));
TS01_07 = double (imread ('./testset01/TS01-07.pgm'));
TS01_08 = double (imread ('./testset01/TS01-08.pgm'));
TS01_09 = double (imread ('./testset01/TS01-09.pgm'));
TS01_10 = double (imread ('./testset01/TS01-10.pgm'));
TS01_11 = double (imread ('./testset01/TS01-11.pgm'));
TS01_12 = double (imread ('./testset01/TS01-12.pgm'));
TS01_13 = double (imread ('./testset01/TS01-13.pgm'));
TS01_14 = double (imread ('./testset01/TS01-14.pgm'));
TS01_15 = double (imread ('./testset01/TS01-15.pgm'));
TS01_16 = double (imread ('./testset01/TS01-16.pgm'));
TS01_17 = double (imread ('./testset01/TS01-17.pgm'));
TS01_18 = double (imread ('./testset01/TS01-18.pgm'));
TS01_19 = double (imread ('./testset01/TS01-19.pgm'));
TS01_20 = double (imread ('./testset01/TS01-20.pgm'));
TS01_21 = double (imread ('./testset01/TS01-21.pgm'));
TS01_22 = double (imread ('./testset01/TS01-22.pgm'));
TS01_23 = double (imread ('./testset01/TS01-23.pgm'));
TS01_24 = double (imread ('./testset01/TS01-24.pgm'));
TS01_25 = double (imread ('./testset01/TS01-25.pgm'));
TS01_26 = double (imread ('./testset01/TS01-26.pgm'));
TS01_27 = double (imread ('./testset01/TS01-27.pgm'));
TS01_28 = double (imread ('./testset01/TS01-28.pgm'));
TS01_29 = double (imread ('./testset01/TS01-29.pgm'));
TS01_30 = double (imread ('./testset01/TS01-30.pgm'));
TS01_31 = double (imread ('./testset01/TS01-31.pgm'));
TS01_32 = double (imread ('./testset01/TS01-32.pgm'));
TS01_33 = double (imread ('./testset01/TS01-33.pgm'));
TS01_34 = double (imread ('./testset01/TS01-34.pgm'));
TS01_35 = double (imread ('./testset01/TS01-35.pgm'));
TS01_36 = double (imread ('./testset01/TS01-36.pgm'));
TS01_37 = double (imread ('./testset01/TS01-37.pgm'));
TS01_38 = double (imread ('./testset01/TS01-38.pgm'));
TS01_39 = double (imread ('./testset01/TS01-39.pgm'));
TS01_40 = double (imread ('./testset01/TS01-40.pgm'));
TS01_41 = double (imread ('./testset01/TS01-41.pgm'));
TS01_42 = double (imread ('./testset01/TS01-42.pgm'));
```

```
[RECTTS01_01, IMRTS01_01] = facescan (NET, TS01_01, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_02, IMRTS01_02] = facescan (NET, TS01_02, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_03, IMRTS01_03] = facescan (NET, TS01_03, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_04, IMRTS01_04] = facescan (NET, TS01_04, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_05, IMRTS01_05] = facescan (NET, TS01_05, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_06, IMRTS01_06] = facescan (NET, TS01_06, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_07, IMRTS01_07] = facescan (NET, TS01_07, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_08, IMRTS01_08] = facescan (NET, TS01_08, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_09, IMRTS01_09] = facescan (NET, TS01_09, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_10, IMRTS01_10] = facescan (NET, TS01_10, MASK, 0.8, 6, 1, 1.2, 2)
fprintf('\n Images 01-10 Complete! \n Press Any Key to Continue \n');
pause;
[RECTTS01_11, IMRTS01_11] = facescan (NET, TS01_11, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_12, IMRTS01_12] = facescan (NET, TS01_12, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_13, IMRTS01_13] = facescan (NET, TS01_13, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_14, IMRTS01_14] = facescan (NET, TS01_14, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_15, IMRTS01_15] = facescan (NET, TS01_15, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_16, IMRTS01_16] = facescan (NET, TS01_16, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_17, IMRTS01_17] = facescan (NET, TS01_17, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_18, IMRTS01_18] = facescan (NET, TS01_18, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_19, IMRTS01_19] = facescan (NET, TS01_19, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_20, IMRTS01_20] = facescan (NET, TS01_20, MASK, 0.8, 6, 1, 1.2, 2)
fprintf('\n Images 10-20 Complete! \n Press Any Key to Continue \n');
pause;
[RECTTS01_21, IMRTS01_21] = facescan (NET, TS01_21, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_22, IMRTS01_22] = facescan (NET, TS01_22, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_23, IMRTS01_23] = facescan (NET, TS01_23, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_24, IMRTS01_24] = facescan (NET, TS01_24, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_25, IMRTS01_25] = facescan (NET, TS01_25, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_26, IMRTS01_26] = facescan (NET, TS01_26, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_27, IMRTS01_27] = facescan (NET, TS01_27, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_28, IMRTS01_28] = facescan (NET, TS01_28, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_29, IMRTS01_29] = facescan (NET, TS01_29, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_30, IMRTS01_30] = facescan (NET, TS01_30, MASK, 0.8, 6, 1, 1.2, 2)
fprintf('\n Images 20-30 Complete! \n Press Any Key to Continue \n');
pause;
[RECTTS01_31, IMRTS01_31] = facescan (NET, TS01_31, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_32, IMRTS01_32] = facescan (NET, TS01_32, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_33, IMRTS01_33] = facescan (NET, TS01_33, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_34, IMRTS01_34] = facescan (NET, TS01_34, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_35, IMRTS01_35] = facescan (NET, TS01_35, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_36, IMRTS01_36] = facescan (NET, TS01_36, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_37, IMRTS01_37] = facescan (NET, TS01_37, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_38, IMRTS01_38] = facescan (NET, TS01_38, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_39, IMRTS01_39] = facescan (NET, TS01_39, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_40, IMRTS01_40] = facescan (NET, TS01_40, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_41, IMRTS01_41] = facescan (NET, TS01_41, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS01_42, IMRTS01_42] = facescan (NET, TS01_42, MASK, 0.8, 6, 1, 1.2, 2)

fprintf('\n\nPerfomance Analysis Complete\n\n');
```

9.2.6 testset02.m

```
% Author: Phil Brimblecombe
% Email:  ee01pb@surrey.ac.uk
% Course: MEng Electronic Engineering
% Desc:   This script was written to test the performance of networks with
%         TestSet02, images from CMU-VASC - Statistics evaluated manually

%LOAD TEST IMAGES
TS02_01 = double (imread ('./testset02/TS02-01.pgm'));
TS02_02 = double (imread ('./testset02/TS02-02.pgm'));
TS02_03 = double (imread ('./testset02/TS02-03.pgm'));
TS02_04 = double (imread ('./testset02/TS02-04.pgm'));
TS02_05 = double (imread ('./testset02/TS02-05.pgm'));
TS02_06 = double (imread ('./testset02/TS02-06.pgm'));
TS02_07 = double (imread ('./testset02/TS02-07.pgm'));
TS02_08 = double (imread ('./testset02/TS02-08.pgm'));
TS02_09 = double (imread ('./testset02/TS02-09.pgm'));
TS02_10 = double (imread ('./testset02/TS02-10.pgm'));

TS02_11 = double (imread ('./testset02/TS02-11.pgm'));
TS02_12 = double (imread ('./testset02/TS02-12.pgm'));
TS02_13 = double (imread ('./testset02/TS02-13.pgm'));
TS02_14 = double (imread ('./testset02/TS02-14.pgm'));
TS02_15 = double (imread ('./testset02/TS02-15.pgm'));
TS02_16 = double (imread ('./testset02/TS02-16.pgm'));
TS02_17 = double (imread ('./testset02/TS02-17.pgm'));
TS02_18 = double (imread ('./testset02/TS02-18.pgm'));
TS02_19 = double (imread ('./testset02/TS02-19.pgm'));
TS02_20 = double (imread ('./testset02/TS02-20.pgm'));

% Scan the pyramid for faces for images 01 - 20
[RECTTS02_01, IMRTS02_01] = facescan (NET, TS02_01, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_02, IMRTS02_02] = facescan (NET, TS02_02, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_03, IMRTS02_03] = facescan (NET, TS02_03, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_04, IMRTS02_04] = facescan (NET, TS02_04, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_05, IMRTS02_05] = facescan (NET, TS02_05, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_06, IMRTS02_06] = facescan (NET, TS02_06, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_07, IMRTS02_07] = facescan (NET, TS02_07, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_08, IMRTS02_08] = facescan (NET, TS02_08, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_09, IMRTS02_09] = facescan (NET, TS02_09, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_10, IMRTS02_10] = facescan (NET, TS02_10, MASK, 0.8, 6, 1, 1.2, 2)

[RECTTS02_11, IMRTS02_11] = facescan (NET, TS02_11, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_12, IMRTS02_12] = facescan (NET, TS02_12, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_13, IMRTS02_13] = facescan (NET, TS02_13, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_14, IMRTS02_14] = facescan (NET, TS02_14, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_15, IMRTS02_15] = facescan (NET, TS02_15, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_16, IMRTS02_16] = facescan (NET, TS02_16, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_17, IMRTS02_17] = facescan (NET, TS02_17, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_18, IMRTS02_18] = facescan (NET, TS02_18, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_19, IMRTS02_19] = facescan (NET, TS02_19, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_20, IMRTS02_20] = facescan (NET, TS02_20, MASK, 0.8, 6, 1, 1.2, 2)

fprintf('\n\n01-20 Complete!\nPress any key to continue:\n\n');
pause;
```

```
%LOAD TEST IMAGES
TS02_21 = double (imread ('./testset02/TS02-21.pgm'));
TS02_22 = double (imread ('./testset02/TS02-22.pgm'));
TS02_23 = double (imread ('./testset02/TS02-23.pgm'));
TS02_24 = double (imread ('./testset02/TS02-24.pgm'));
TS02_25 = double (imread ('./testset02/TS02-25.pgm'));
TS02_26 = double (imread ('./testset02/TS02-26.pgm'));
TS02_27 = double (imread ('./testset02/TS02-27.pgm'));
TS02_28 = double (imread ('./testset02/TS02-28.pgm'));
TS02_29 = double (imread ('./testset02/TS02-29.pgm'));
TS02_30 = double (imread ('./testset02/TS02-30.pgm'));

TS02_31 = double (imread ('./testset02/TS02-31.pgm'));
TS02_32 = double (imread ('./testset02/TS02-32.pgm'));
TS02_33 = double (imread ('./testset02/TS02-33.pgm'));
TS02_34 = double (imread ('./testset02/TS02-34.pgm'));
TS02_35 = double (imread ('./testset02/TS02-35.pgm'));
TS02_36 = double (imread ('./testset02/TS02-36.pgm'));
TS02_37 = double (imread ('./testset02/TS02-37.pgm'));
TS02_38 = double (imread ('./testset02/TS02-38.pgm'));
TS02_39 = double (imread ('./testset02/TS02-39.pgm'));
TS02_40 = double (imread ('./testset02/TS02-40.pgm'));

TS02_41 = double (imread ('./testset02/TS02-41.pgm'));
TS02_42 = double (imread ('./testset02/TS02-42.pgm'));
TS02_43 = double (imread ('./testset02/TS02-43.pgm'));
TS02_44 = double (imread ('./testset02/TS02-44.pgm'));
TS02_45 = double (imread ('./testset02/TS02-45.pgm'));
TS02_46 = double (imread ('./testset02/TS02-46.pgm'));
TS02_47 = double (imread ('./testset02/TS02-47.pgm'));
TS02_48 = double (imread ('./testset02/TS02-48.pgm'));
TS02_49 = double (imread ('./testset02/TS02-49.pgm'));
TS02_50 = double (imread ('./testset02/TS02-50.pgm'));

% Scan the pyramid for faces for images 21-50
[RECTTS02_21, IMRTS02_21] = facescan (NET, TS02_21, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_22, IMRTS02_22] = facescan (NET, TS02_22, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_23, IMRTS02_23] = facescan (NET, TS02_23, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_24, IMRTS02_24] = facescan (NET, TS02_24, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_25, IMRTS02_25] = facescan (NET, TS02_25, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_26, IMRTS02_26] = facescan (NET, TS02_26, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_27, IMRTS02_27] = facescan (NET, TS02_27, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_28, IMRTS02_28] = facescan (NET, TS02_28, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_29, IMRTS02_29] = facescan (NET, TS02_29, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_30, IMRTS02_30] = facescan (NET, TS02_30, MASK, 0.8, 6, 1, 1.2, 2)

[RECTTS02_31, IMRTS02_31] = facescan (NET, TS02_31, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_32, IMRTS02_32] = facescan (NET, TS02_32, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_33, IMRTS02_33] = facescan (NET, TS02_33, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_34, IMRTS02_34] = facescan (NET, TS02_34, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_35, IMRTS02_35] = facescan (NET, TS02_35, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_36, IMRTS02_36] = facescan (NET, TS02_36, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_37, IMRTS02_37] = facescan (NET, TS02_37, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_38, IMRTS02_38] = facescan (NET, TS02_38, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_39, IMRTS02_39] = facescan (NET, TS02_39, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_40, IMRTS02_40] = facescan (NET, TS02_40, MASK, 0.8, 6, 1, 1.2, 2)

[RECTTS02_41, IMRTS02_41] = facescan (NET, TS02_41, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_42, IMRTS02_42] = facescan (NET, TS02_42, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_43, IMRTS02_43] = facescan (NET, TS02_43, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_44, IMRTS02_44] = facescan (NET, TS02_44, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_45, IMRTS02_45] = facescan (NET, TS02_45, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_46, IMRTS02_46] = facescan (NET, TS02_46, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_47, IMRTS02_47] = facescan (NET, TS02_47, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_48, IMRTS02_48] = facescan (NET, TS02_48, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_49, IMRTS02_49] = facescan (NET, TS02_49, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_50, IMRTS02_50] = facescan (NET, TS02_50, MASK, 0.8, 6, 1, 1.2, 2)

fprintf('\n\n20-50 Complete!\nPress any key to continue:\n\n');
pause;
```

```
%LOAD TEST IMAGES
TS02_51 = double (imread ('./testset02/TS02-51.pgm'));
TS02_52 = double (imread ('./testset02/TS02-52.pgm'));
TS02_53 = double (imread ('./testset02/TS02-53.pgm'));
TS02_54 = double (imread ('./testset02/TS02-54.pgm'));
TS02_55 = double (imread ('./testset02/TS02-55.pgm'));
TS02_56 = double (imread ('./testset02/TS02-56.pgm'));
TS02_57 = double (imread ('./testset02/TS02-57.pgm'));
TS02_58 = double (imread ('./testset02/TS02-58.pgm'));
TS02_59 = double (imread ('./testset02/TS02-59.pgm'));
TS02_60 = double (imread ('./testset02/TS02-60.pgm'));

TS02_61 = double (imread ('./testset02/TS02-61.pgm'));
TS02_62 = double (imread ('./testset02/TS02-62.pgm'));
TS02_63 = double (imread ('./testset02/TS02-63.pgm'));
TS02_64 = double (imread ('./testset02/TS02-64.pgm'));
TS02_65 = double (imread ('./testset02/TS02-65.pgm'));


% Scan the pyramid for faces for images 51-65
[RECTTS02_51, IMRTS02_51] = facescan (NET, TS02_51, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_52, IMRTS02_52] = facescan (NET, TS02_52, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_53, IMRTS02_53] = facescan (NET, TS02_53, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_54, IMRTS02_54] = facescan (NET, TS02_54, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_55, IMRTS02_55] = facescan (NET, TS02_55, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_56, IMRTS02_56] = facescan (NET, TS02_56, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_57, IMRTS02_57] = facescan (NET, TS02_57, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_58, IMRTS02_58] = facescan (NET, TS02_58, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_59, IMRTS02_59] = facescan (NET, TS02_59, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_60, IMRTS02_60] = facescan (NET, TS02_60, MASK, 0.8, 6, 1, 1.2, 2)

[RECTTS02_61, IMRTS02_61] = facescan (NET, TS02_61, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_62, IMRTS02_62] = facescan (NET, TS02_62, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_63, IMRTS02_63] = facescan (NET, TS02_63, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_64, IMRTS02_64] = facescan (NET, TS02_64, MASK, 0.8, 6, 1, 1.2, 2)
[RECTTS02_65, IMRTS02_65] = facescan (NET, TS02_65, MASK, 0.8, 6, 1, 1.2, 2)

fprintf('\n\nPerfomance Analysis Complete\n\n');
```

9.3 Examples of Classifier Performance

9.3.1 Test Set Summary

TESTSET01

Image Name	Resolution	Comments
TS01-01	392 x 272	
TS01-02	302 x 204	Poor quality
TS01-03	462 x 294	Skin Colour – not in training set
TS01-04	852 x 568	Adverse Lighting Conditions
TS01-05	277 x 801	Includes text
TS01-06	320 x 240	TV broadcast
TS01-07	320 x 240	TV broadcast
TS01-08	320 x 240	TV broadcast
TS01-09	320 x 240	TV broadcast
TS01-10	320 x 240	TV broadcast
TS01-11	320 x 240	TV broadcast
TS01-12	320 x 240	TV broadcast
TS01-13	320 x 240	TV broadcast
TS01-14	630 x 418	Skin Colour – not in training set
TS01-15	268 x 414	Glasses and occlusion
TS01-16	814 x 820	Some rotation
TS01-17	126 x 210	Skin Colour – not in training set
TS01-18	640 x 480	Includes sketches of faces
TS01-19	716 x 684	Rotation
TS01-20	268 x 406	Rotation
TS01-21	274 x 406	
TS01-22	414 x 273	Rotation
TS01-23	320 x 240	TV broadcast
TS01-24	320 x 240	TV broadcast
TS01-25	320 x 240	TV broadcast
TS01-26	320 x 240	TV broadcast
TS01-27	144 x 192	TV broadcast Skin Colour – not in training set
TS01-28	320 x 240	TV broadcast
TS01-29	320 x 240	TV broadcast
TS01-30	276 x 343	TV broadcast
TS01-31	640 x 422	Adverse Lighting Conditions
TS01-32	418 x 258	Dark + Skin Colour – not in training set
TS01-33	418 x 255	Skin Colour – not in training set
TS01-34	662 x 422	
TS01-35	834 x 426	Rotation
TS01-36	696 x 510	Skin Colour – not in training set
TS01-37	539 x 734	Includes text
TS01-38	580 x 380	Skin Colour – not in training set
TS01-39	412 x 273	
TS01-40	271 x 403	
TS01-41	468 x 720	
TS01-42	623 x 805	

TESTSET02

Image Name	Resolution	Comments
TS02-01	864 x 890	
TS02-02	576 x 776	Cartoon
TS02-03	344 x 499	
TS02-04	256 x 377	Large Face – too big for standard image pyramid
TS02-05	280 x 484	
TS02-06	358 x 350	
TS02-07	522 x 460	
TS02-08	610 x 395	
TS02-09	60 x 75	Low resolution
TS02-10	177 x 215	Glasses
TS02-11	564 x 388	Complex Scenery
TS02-12	450 x 672	Rotated and partially occluded
TS02-13	1154 x 678	Some glasses, some occluded by others or hair
TS02-14	396 x 552	Complex Scenery
TS02-15	439 x 300	
TS02-16	580 x 861	
TS02-17	592 x 843	
TS02-18	500 x 500	Skin Colour – not in training set + rotated faces
TS02-19	311 x 350	
TS02-20	340 x 350	Rotation
TS02-21	468 x 464	
TS02-22	552 x 388	
TS02-23	328 x 430	
TS02-24	159 x 160	
TS02-25	392 x 584	Complex Scenery
TS02-26	660 x 656	
TS02-27	715 x 486	Skin Colour – not in training set
TS02-28	320 x 256	Rotation
TS02-29	352 x 352	Skin Colour – not in training set
TS02-30	336 x 484	Glasses
TS02-31	500 x 500	Skin Colour – not in training set + rotated faces
TS02-32	345 x 348	
TS02-33	824 x 956	Includes text
TS02-34	1168 x 832	Includes text
TS02-35	500 x 622	
TS02-36	360 x 259	
TS02-37	813 x 1074	Cartoon + Sketch
TS02-38	520 x 739	Mona Lisa - rotated
TS02-39	592 x 654	Partial occlusion and too large for image pyramid
TS02-40	256 x 256	
TS02-41	500 x 500	Rotation
TS02-42	696 x 1056	Playing Cards
TS02-43	323 x 229	Rotation
TS02-44	384 x 592	Complex Scenery
TS02-45	555 x 768	Includes text
TS02-46	580 x 396	Skin Colour – not in training set
TS02-47	476 x 471	Large Face – too big for standard image pyramid
TS02-48	360 x 360	
TS02-49	469 x 375	
TS02-50	108 x 144	
TS02-51	384 x 556	Complex Scenery
TS02-52	367 x 364	
TS02-53	535 x 630	Rotation
TS02-54	490 x 338	
TS02-55	400 x 584	Complex Scenery
TS02-56	348 x 352	
TS02-57	336 x 344	
TS02-58	468 x 464	Complex non-face
TS02-59	580 x 392	
TS02-60	340 x 484	
TS02-61	443 x 599	Glasses
TS02-62	289 x 158	Occluded
TS02-63	97 x 101	Low resolution
TS02-64	812 x 1060	Text – Phonebook
TS02-65	200 x 190	

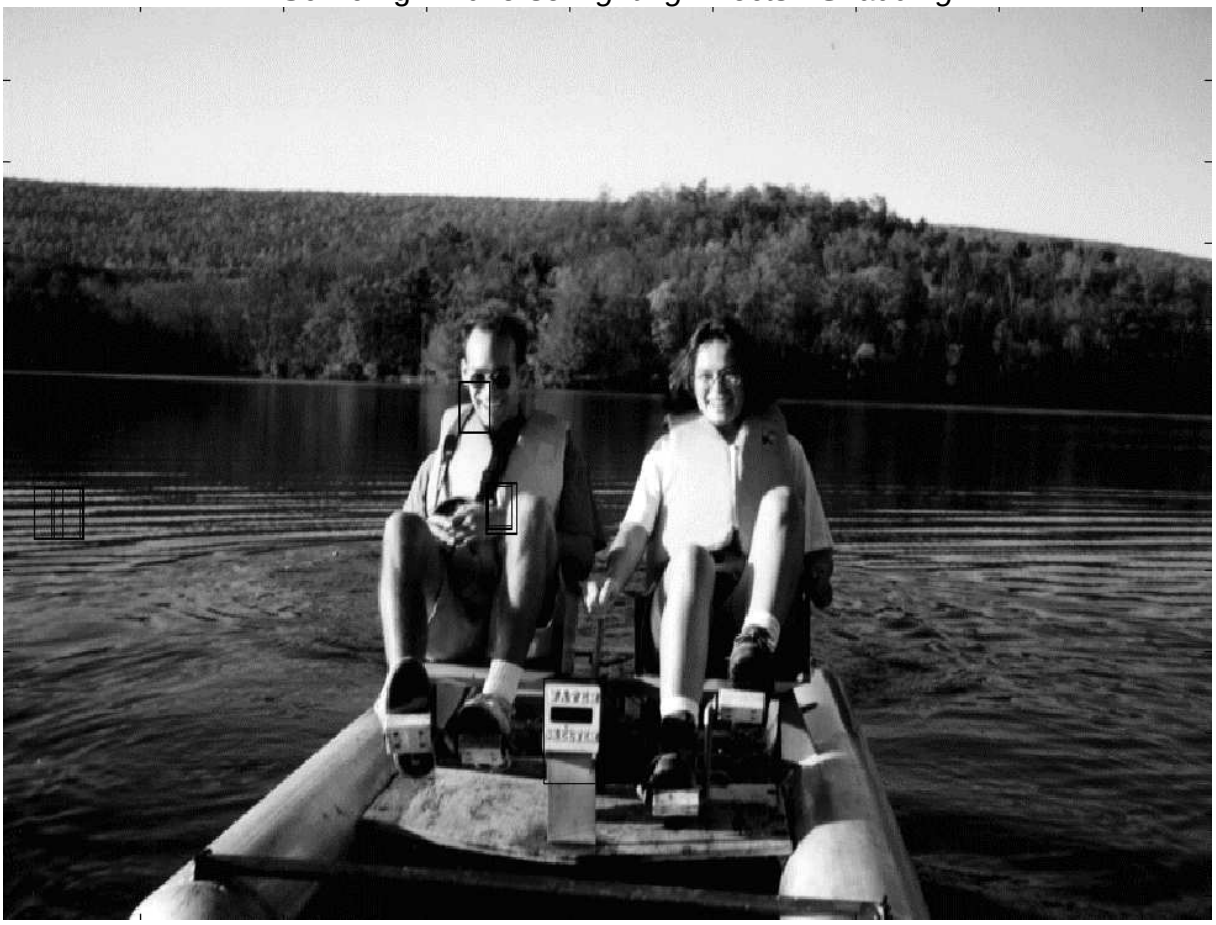
9.3.2 Original Detector

TS01-32.gif: *Missed faces due to skin colour and moustache – not in training set.*

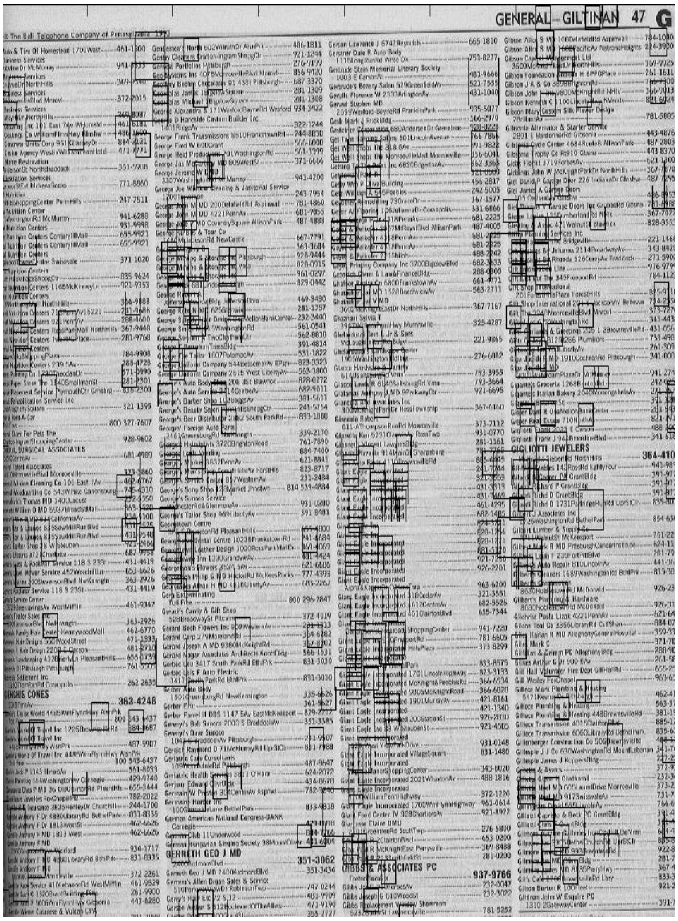
TS02-02.gif: *Cartoon Face – no similar examples in training set*

TS02-04.gif: *Large face – too big for image pyramid*

TS01-04.gif: *Adverse Lighting Effects - Shadding*



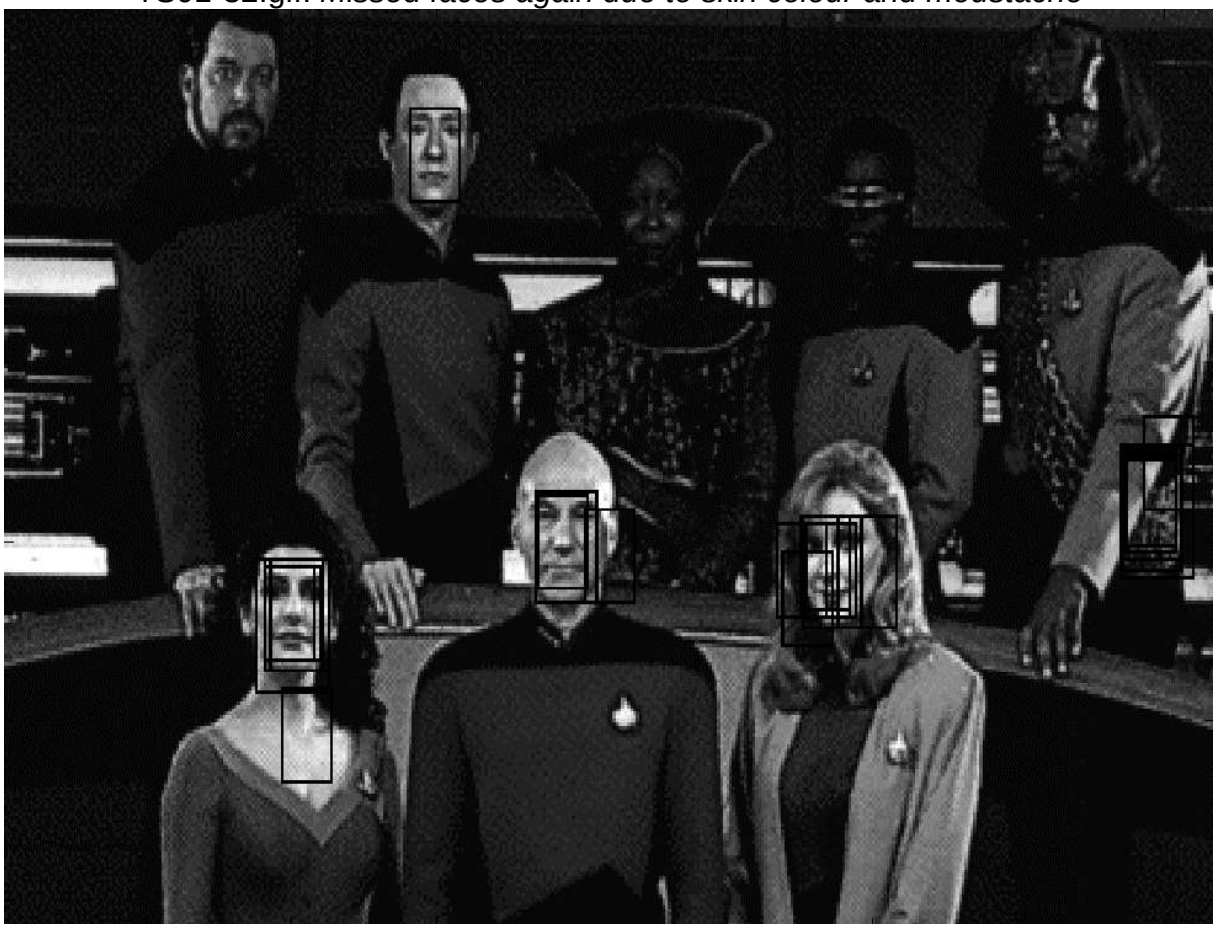
TS02-55.gif / TS02-64.gif: *Very Complex Scenery Examples*



TS02-55.gif: *Mulitple faces – some occlusion, some bespectacled faces*

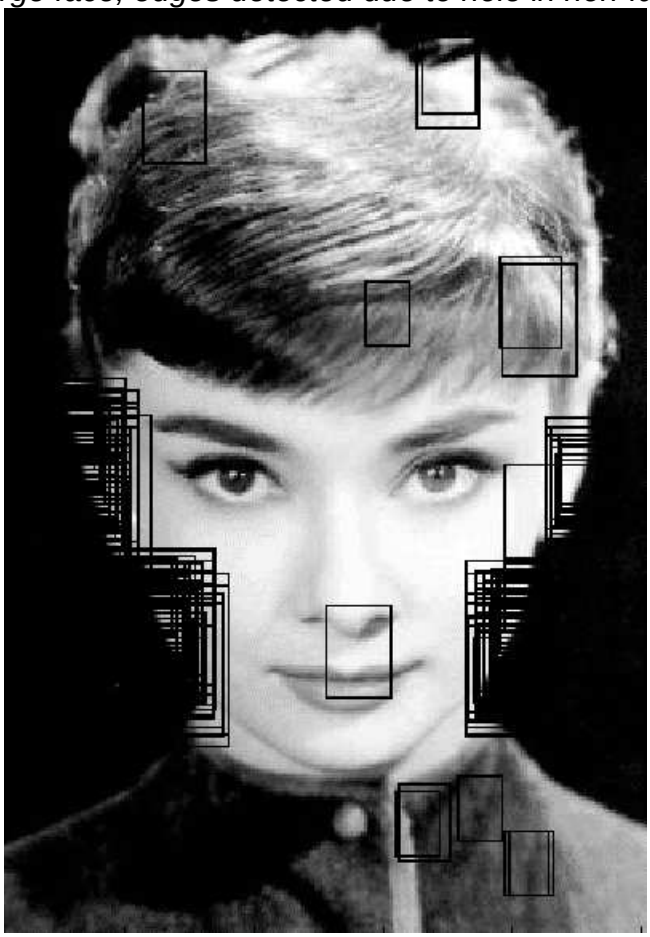
9.3.3 Final Improved Detector

TS01-32.gif: *Missed faces again due to skin colour and moustache*

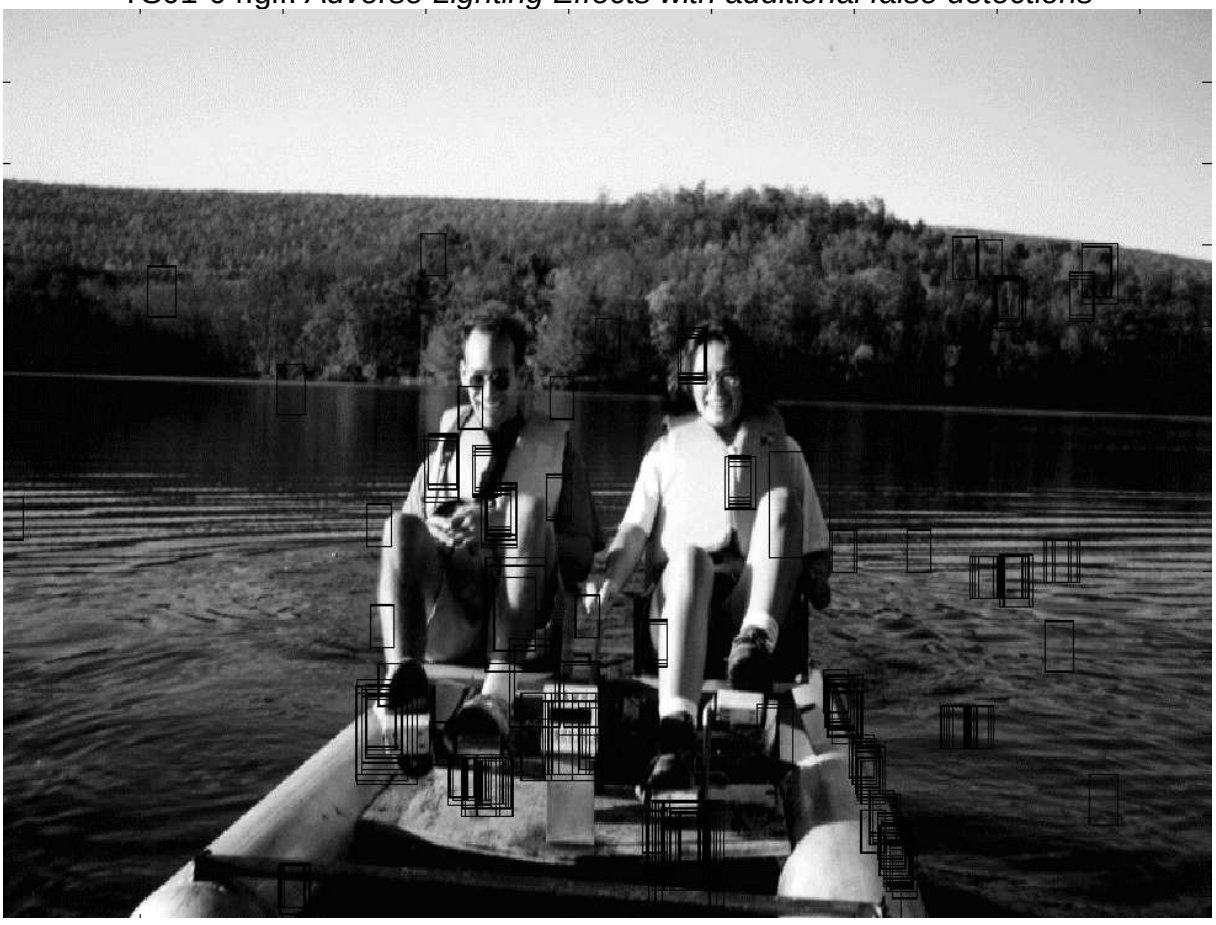


TS02-02.gif: *Cartoon Face – unacceptable number of false positives*

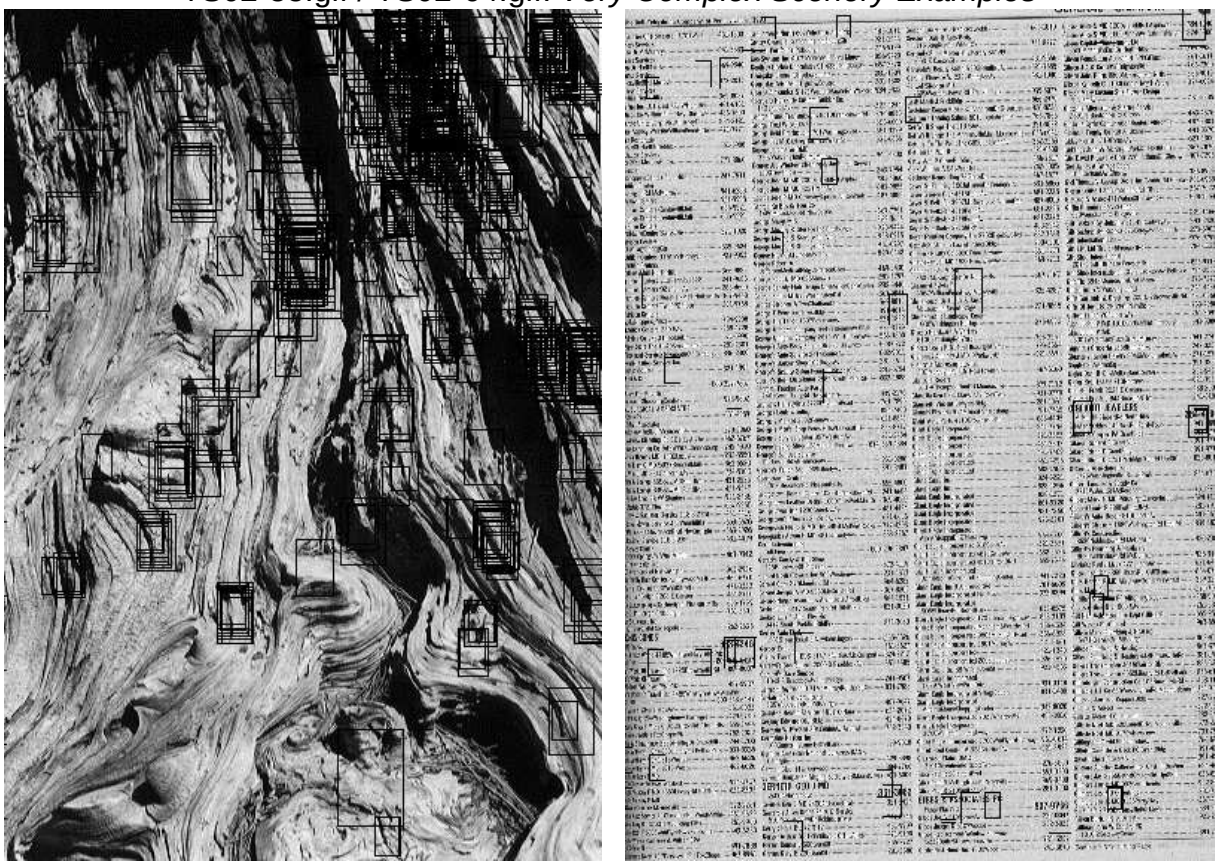
TS02-04.gif: *Large face, edges detected due to hole in non-face example set*



TS01-04.gif: *Adverse Lighting Effects with additional false detections*



TS02-55.gif / TS02-64.gif: *Very Complex Scenery Examples*



TS02-13.gif: *Mulitple faces – some occlusion, some bespectacled faces*