UNIVERSITETET I OSLO
Institutt for informatikk

Implementing an automatic face recognition system

Hovedoppgave

Nikolai Czajkowski

1. august 2006

Summary

The goal of this project was to *implement* and *evaluate* an automatic face recognition system. Face localization was adapted from the algorithm of Rowley et.al. A 20×20 pixel sliding window scans a pyramid of the input image, feeding into several neural networks. The router network estimates rotation, while three detector networks classify the window as a face or non-face. This approach is very computationally intensive, and estimating rotation is the most demanding aspect. Our contribution to the localization algorithm is to reduce this processing time by i) using a genetic algorithm to evolve a more sparsely connected router network, and ii) to limit the search space by thresholding the input image on skin colour and texture. An eigenface based algorithm was used for face recognition. Our contribution was to have the system automatically extract multiple and differently segmented faces from the training image, rather than manually segmenting a single face.

Evaluation of the localization algorithm demonstrated that it is generally robust, able to correctly and accurately segment faces despite variations in lighting, rotation, scale, facial expression and occlusion. Furthermore, although the difference between network topologies was small, the evolved router topology improved speed by roughly 13%, while improving the localization rate compared to Rowleys fully connected topology by 3%. Texture thresholding reduced the running time of the algorithm by almost two thirds in both colour and grayscale images, while skin colour thresholding independently reduced the running time by 45%. In colour images the combined effect was a reduction in processing time of 75%.

Benchmarking of the recognition algorithm demonstrated that eigenface-based recognition is very sensitive to variations in face segmentation relative to the stored template. However, because automatic segmentation is likely to vary somewhat compared to a manual segmentation of the same face, performance of the eigenface based approach can be improved by automatically extracting and storing several slightly differently segmented templates per person.

The program was implemented in C, and new libraries for neural networks, image processing, TIFF reading/writing and genetic algorithms were written to make the system fast and flexible.

Acknowledgments

I would like to thank my supervisor Anne Solberg for a patience that has now spanned a considerable number of semesters, and for much helpful advice.

I would also like to thank my friends, family, and girlfriend. I'm looking forward to seeing you all again.

Oslo, July 2006
Nikolai Czajkowski

Contents

1 Introduction

In 2001 the city of Tampa, Florida hosted the annual American Football league final, the Super Bowl. That year, in addition to the regular media frenzy, the finals received an extra dose of media attention as city officials had installed an automatic face recognition system, "FaceIt", in the stadium entrance. Cameras recorded people entering, "FaceIt" localized their faces in the video stream and compared them to images in a police database. The initiative slingshot biometric systems into public awareness, and earned the city the "Worst public official" award from Privacy International. Similar systems have since been installed in a handful of locations, but so far they have not led to any arrests. The lack of arrests highlights the sobering fact that despite the recent upsurge in public imagination and availability of commercial face recognition products, this technology has yet to mature, and robust recognition remains a challenge.

Perhaps here more than in any other area of machine vision, the relative ease with which people perform the task compared to computers becomes apparent. A frontal face in your field of vision potentially indicates that an individual has its attention directed at you, and this could be an important event. Also, for more social animals like us, it is vital to be able to reliably distinguish between the members of our group. As vision is our primary sense, it is natural that this discrimination is based on visual perception. Faces are sufficiently complex to contain the necessary information to distinguish individuals, and even contain information about the person's mood. As face recognition is so important for us, many have speculated that specialized neural structures have evolved to facilitate this task. The degree to which people are endowed with such specialized "mental modules" is still heatedly debated, and we will briefly return to it later. What is certain however, is that regardless of how specific the neural circuits are, already shortly after birth children are oriented towards face-like visual stimuli [27]. By face-like we refer to patterns characterized by symmetry and a certain degree of complexity. So, from the very beginning and throughout life, individuals are exposed to a massive amount of faces. Combined with possible "hard wired" neural circuits, this experience results in faces being the class of stimuli where we have the finest level of discriminability. As with many tasks in pattern recognition, what seems almost effortless to humans has proved very difficult to implement robustly in a computer, even though automatic face detection and recognition has been an active area of research for almost 30 years [42].

The thesis begins with a short repetition of some of the basic technologies used in the algorithm, such as colour space, neural networks and genetic algorithms. I assume the reader is familiar with these subjects already, but a short introduction will later make it easier to discuss their purpose in the implemented system. Any system that attempts to recognize a face, or try to respond to the presence of one in some way, must first find it in the image. In practice this involves segmenting the face from the background by finding as exactly as possible the bounding coordinates. While localization and recognition may be intertwined in human face recognition, it presents a natural divide in automated algorithms, and I have chosen to present these steps of the algorithm separately in chapters 3 and 4. Chapter 5 gives a summary of how the ideas outlined in the previous chapters are implemented into a working system, as well as presenting some major design decisions. In chapter 6 results from benchmarking of the system are presented. In chapter 7 I reflect on some of the more general lessons learned from implementing the system the way that we did.

1.1 Biometrics

> Biometrics (ancient Greek: bios ="life", metron ="measure") is the study of automated methods for uniquely recognizing humans based upon one or more intrinsic physical or behavioral traits [36].

The definition given above contains two criteria that must be fulfilled by any biometric system. It must be *automated* in the acquisition of data, the extraction of relevant features, and the classification. Secondly, the features where by this classification is performed are *physical or behavioral*. These features can be acquired from one or more physical domains, such as fingerprints, iris, voice, face, body shape, weight etc. How the system subsequently processes the data depends on what kind of data it has collected, as well as the purpose of the system. In terms of purpose, it is common to differentiate between identification and identity verification. An *identification* application such as the FaceIt system introduced above, attempts to determine the identity of a given individual. On the other hand, an *identity verification* system attempts to verify that the individual really is who he/she claims to be.

Before proceeding, let us refresh some fundamental concepts in classification theory. In face localization, the task of the classifier is to determine whether a feature vector represents a face. The classifier can categorize the input correctly,

either *true positive* if a face is present, or *true negative* if it isn't. If the classifier makes an error it can either be by indicating a face is present when it in fact is not, *false positive*, or claiming that it is not present when it is, *false negative*.

Often there is a trade-off between the degree of certainty of a biometric method and how invasive it is in acquiring the necessary information. DNA analysis, which is becoming increasingly automated, and can under certain circumstances be regarded as a biometric system, will identify a person with virtually no level of error. However the procedure requires a physical sample of the individual. So, biometric systems differ in many respects, how do we compare alternative approaches? Biometric systems can be compared on the following criteria [14]:

- *Universality:* to what degree everyone posses the necessary features.
- *Uniqueness:* whether the physical features uniquely distinguish between different individuals.
- *Permanence:* whether the physical features used are invariant over time, or can be altered by the individual.
- *Acceptability:* to what degree people will accept being subjected to analysis.
- *Collectability:* how easy it is to collect the necessary data.
- *Performance:* how well the best algorithms perform in terms of speed, accuracy and robustness.
- *Circumvention:* how easy it is to fool the system.

As an example, take fingerprints. They have fairly high universality, although scarring can interfere with classification, and an estimated 5% of the population does not have elligable prints [8]. Fingerprints are to such an extent unique that even identical twins differ, as do the prints of the same individual across fingers. Furthermore, fingerprint vary little over time, and current biometric systems can use them to classify individuals with very low levels of error. However, fingerprints have some acceptance issues, as they are usually associated with criminal investigations. Also, in collecting the data you need the individuals cooperation, or at the very least, awareness and attention. On the other hand, face recognition can be carried out from a distance, even without the individual being aware. So universality, acceptability and collectability of face recognition systems are high. However, faces

though theoretically unique, are perhaps objectively more similar than we think. Furthermore, they vary over time as people wear glasses, grow beards or put on makeup. The performance of state of the art algorithms is therefore still fairly low.

Simplified, any biometric system consists of two major components. First features must be extracted, then they must be compared with templates in a database. Once data of a certain physical domain has been sampled, and before the classification can be carried out, there is usually some preprocessing required. When a fingerprint scanner gathers data, some translation or rotation may be necessary, but otherwise the data is fairly well segmented. This is not the case for most face recognition applications. Their input is usually a two dimensional image, and the first task of the system will invariably be to find as accurately as possible the coordinates of any faces present. Finding faces in images may seem trivial, after all, we all know what they look like. However, I will throughout this thesis attempt to demonstrate that localization is in fact a challenging task, at least as the visual scene the individual is in becomes more complex and the lighting conditions less ideal.

Some of the factors complicating the task of face localization will be discussed and illustrated in section 3.2. For now it is suffice to say that the shape of the face can cause two-dimensional representations to vary considerably when light is cast from different angles. Furthermore, occlusion by objects such as cigarettes, glasses and in turn their shadows makes detection still more difficult. Also, faces can be at any distance or orientation relative to the camera.

If localization is such a challenge, is it worth the effort? Why would we need to localize people in images? The most obvious and perhaps most profiled use of this technology is surveillance and security. Given the escalating concerns with airport security and border controls, the technology has at least in these sectors enormous economic potential. But there are other and perhaps more benign uses for the same technology that receive less attention. With the steep growth in sales of digital cameras, the integration of increasingly high quality digital cameras in embedded electronics such as mobile phones and PDA's, digitization of video archives, proliferation of broadband access, vasts amounts of video and images are becoming available. Yet if this rise in availability is to result in a genuine increase in accessibility, then each image must be labeled and tagged with content information for retrieval and searching. Unless such textual information regarding content is stored along with this data, it will increasingly be rendered inaccessible, lost in a jungle of information. The sheer amount of images to search manually becomes prohibitive.

This is part of the reason for the growing interest in *content based indexing (CBI)*. In CBI, a database is indexed not as conventionally done through a tag, verbal descriptor or elementary data type, but by the actual contents of the multimedia item. In other words, rather than having a tag describing the contents along with the data, ex. "car", the information is referenced through features extracted from the data itself. The reason why face localization and recognition algorithms are particularly relevant to CBI, is that automatic face recognition is one of few attempts to classify objects that can vary substantially over time and situations, but where the differences between the individual class members can be very small. I.e. a person can change facial expression, grow a beard or wear glasses, but the difference between individual people is relatively subtle. Face recognition is therefore particularly difficult, compared to automated recognition of rigid objects such as cars or airplanes. Also, because the relevant features needed to differentiate individual faces can change under different conditions, most of the proposed algorithms are able to learn from a set of training examples. They are therefore sufficiently generic to be applied to localization or recognition of most other classes of objects, given a different training set.

Human machine interaction is the final use for face localization/recognition algorithms I will mention, and it is one which is likely to increase in importance in the near future. Faces are not just any class of stimuli. We expect feedback to be directed to our faces, and our expressions also convey a lot of information beyond our identity. Therefore, as human-computer interaction becomes more widespread, the need for robust ways to detect, recognize and interpret the nonverbal information present in faces will become more important.

2 Introduction to the constituent parts of the algorithm

2.1 Face perception, brain and biology; what can be learned, and what should be used?

Engineers are often inspired by the elegance and functionality of biological solution to difficult problems. However, as they are working with very different raw materials, on vastly different time scales and levels of complexity, it is not given that biological solutions can or should be applied to any given engineering problem. However, as will be evident throughout this thesis, several aspects of the presented algorithm are inspired by biological systems and processes. Also, as face recognition is a skill all healthy children develop seemingly without effort, it might be useful to sketch how this is accomplished by the human brain. This section therefore gives a very brief introduction to the functional neurobiology of perception, before describing some relevant and important findings in neuropsychology.

We can define *receptive field* of a neuron as the region of the visual field in which stimuli will affect its level of activity. As one moves up the neural hierarchy, from the sensory cells to association areas of the neocortex[1], the receptive fields of the cells become larger, and the stimuli that elict their response becomes increasingly complex and abstract [5].

The neural processes of recognition begin in the retina, where light focused through the lens hits photo-pigment molecules in the specialized cells lining the wall, causing these molecules to break down. The constituent parts of the breakdown in turn cause ion channels [2] to change shape, allowing an influx of Na^+, ultimately resulting in an electrical signal. However, the retina does more than simply transform light to an electrochemical analogue. A large amount of low level signal processing is performed in the retina. Information is carried from the eyes through the optic nerve, which is formed of special cells called *ganglion cells*. Due to processing in the retina, activity of ganglion cells does not merely represent the presence of light at a particular point in our field of vision. Rather, these cells respond maximally to differences in intensity between the center and the surrounding in their receptive field. This way retinal ganglion cells convey information about discontinuities in

[1] phylogenetically newer regions of the brain that are neither motor or sensory but are thought to be involved in higher processing of information.

[2] complex molecules in the cell membrane

light intensity, and these discontinuities often represent the edges of objects.

Through the optic nerve, information is relayed through parts of the mid-brain to the primary *visual cortex*. Still, in the region that the optic nerve feeds into the cortex, the firing of the neurons represent very basic and fairly understandable aspects of the visual stimuli. However, as one moves away into the surrounding associative cortex, more abstract and integrative qualities of the visual stimulus is processed (as is true for all other primary sensory cortical areas).

However, one of the broad distinctions usually made in visual processing is that of the *ventral* and *dorsal stream*[3]. The ventral stream passes information to the temporal lobe, and is mostly concerned with shape (object) recognition. The dorsal stream follows a path up toward the parietal lobe, and is mostly responsible for localizing objects in space. As we follow the dorsal stream further towards the motor areas of the brain, the neural tissue gradually processes our *reaction* to objects, such as reaching for them. These two aspects, localization and recognition, can therefore be dissociated if one of the areas is damaged. A person may recognize an object, but is unable to quickly and accurately reach for it, or vica versa.

The introduction given to neural processing of visual information is of course only an abstraction. The optic nerve is made up of roughly ten times as many fibers as the cochlear nerve [4], and a large part of the cortex and many subcortical areas are in some way involved in the processing of visual stimuli. The structure of the cortex is far more homogeneous than the phylogenetically older neural areas. Therefore, purely structural and physiological knowledge gives limited understanding of how and what information is processed.

Most knowledge of normal brain *functioning* in humans comes not from knowing its structure, but from studying damaged brains. Broadly speaking the three main sources of such damage is shrapnel from war wounds, strokes and tumors. However, comared to what we known about damage to motor areas in the parietal lobe, much less is known about damage to the higher visual areas. The blood vessels that feed the parietal motor areas are much more convoluted and vulnerable for rupture than those of the occipital lobe. Shrapnel damage to the visual areas (occipital lobe) is also likely to damage brain stem areas controlling breathing and blood pressure, and result in fatality.

Damage to the eyes will of course result in blindness, and the same is true of

[3]Ventral and dorsal are medical terms indicating "towards the front" and "towards the top and back", respectively

[4]auditory nerve

damage to the optic nerve or primary visual cortex. However, damage to areas higher in the processing hierarchy can result in loss of higher level functions, such as colour blindness or inability to perceive movement. *Agnosia* is a term referring to the loss of ability to interpret the stimuli in a given sensory modality. For example, a patient with *visual agnosia* can experience all the constituent elements of perception, such as colour, texture, edges, movement etc., but is unable to integrate them all into a meaningful whole. One rare but highly studied version of this disorder is that of *prosopagnosia*, thought to be a selective impairment in recognizing faces resulting from damage to certain areas of the ventral stream. A patient is often able to judge correctly the expression of faces and to discriminate between instances of other classes of objects such vegetables and fruit. Prosopagnosia is among the evidence often cited as indicating dedicated neural circuits evolved for face recognition. However, just how selective the impairment is is a matter of some controversy [7], as most patients have reduced performance on many tasks requiring subtle visual within-category discrimination. Face recognition is perhaps the most complex visual task people routinely perform, and suddenly failing to recognize their spouse or children is certainly the most noticeable and debilitating result of the brain damage. For a further discussion see [2] or [13].

While much is known about the low-level visual system, neurophysiology has not significantly guided face recognition algorithms beyond providing a biological justification for using gabor filters and feed-forward neural networks[5] [26] . But although traditional neural networks are inspired by genuine neurons, they represent only a very simplified model. However, more biologically grounded approaches that mimic the biology of low-level vision have been published [26]. So, what of the higher order properties? One important question that could impact the design of our algorithm is to what degree face perception relies on individual facial features or an holistic evaluation. Do features as eyes, nose and mouth *independently* contribute to recognition, or is face recognition performed on the facial area as a whole, without breaking it down into features. This question is important, as it mirrors two different strategies in biometric systems. We will later describe the eigenface approach that we have implemented, and this approach does not extract features from faces, but simply compares two image areas.

Lastly it should be mentioned that the brain uses all information available, ranging from other sensory modalities to the context in general. Faces are localized faster if they are in a familiar context, and recognition is better if we can hear the per-

[5]During the first few milliseconds, visual processing is thought to be feed-forward.

sons voice, observe typical clothing or see their gait or movement. For this reason multimodal biometric systems[6] has superior performance compared to conventional systems, and is gaining popularity, [16].

2.2 Colour and colour spaces

A *colour space*, is a mathematical way of representing colours as data, typically as three or four values or color components [34]. Many different colour spaces are defined, and the one to use for a given task depends on the purpose of the system. This is conveniently illustrated with the RGB colour model, arguably the most familiar one to most computer scientists. In traditional CRT screens colour was generated by energizing tiny dots of red green and blue coloured phosphorous with an electron beam. These three colours in turn stimulate three separate types of cells in the human retina, and different colours can be generated by mixing the intensities of these basic colours. For the purpose of displaying an image to screen, an RGB colour model may be the optimal one to encode the data, but for other tasks an alternative colour space may be better suited. For example, in RGB, light intensity is not independent of hue. Rather, intensity is the mean value of all three colour dimensions. If we regard say a red object in a brightly lit room, and then dim the lights, the object arguably has the same colour it only reflect less light. Several colour spaces are defined so that intensity is independent of chrominance, as in the HSV/HSI colour space. In HSI I stands for *intensity*, S represents *saturation* (or how vivid the colour is), while H *hue* encodes the dominant wavelength. Colour is important in face localization software, because if it is available it can help us concentrate our search on skin coloured areas. Furthermore, HSI is the colour space most often used in face recognition systems, as studies have shown that difference in the perceived skin colour between individuals of different ethnicity lies mostly in difference in intensity rather than chrominance [19]. It would of course be possible to threshold the skin colour regions in RBG space, but the region constituting skin colour may represent a complex volume or union of such in 3D space. If we transform the colour space in such a way that intensity represents an independent dimension, we can threshold the 2D hue/saturation plane instead. Also, even though there are efficient algorithms for thresholding skin colour regions directly from RBG colour space, empirical evaluations have found alternative ones (HSI and YCbCr) superior [28].

[6] systems that use more than one biometric feature

Computer images are however usually encoded in RGB, so the first step is usually to transform them to HSI colour space. The following simple formulas transforms RGB into to HSI space. Here {R,G,B} represent the three bands in the input image, and {I,S,V} in bands in the transformed image.

$$
\begin{aligned}
I &= \frac{R+B+G}{3} \\
S &= \frac{max(R,G,B) - min(R,G,B)}{max(R,G,B)} \\
H &= \left[acos\left(\frac{.5 \times ((R-G)+(R-B))}{((R-G)*(R-G))+((R-B)*(G-B))} \right) \right] \times \frac{360}{2\pi}
\end{aligned}
$$

Many face localization approaches has skin colour thresholding as an initial step in the processing, either directly to find faces, or more indirectly to reduce the search space. This is useful, as the property is fast to evaluate, invariant of rotation and scaling, and is fairly insensitive to changes in light intensity.

2.3 Neural networks

A biological neural network is an interconnected group of neurons. The term artificial neural network *(ANN)* is used for mathematical models that mimic some of the fundamental characteristics of the way information is represented and processed in biological neural networks.

Before going on to describe in some detail the elements and characteristics of ANNs, I'll briefly review the history of their development. Although *cognitive neuroscience*[7] is a very active field of research, most aspects of the encoding and processing of information in biological neural systems is yet to be fully understood. In the 1940's, the synapse, and the chemical transmission in the synapse was a recent discovery, and gave rise to the first models of how the physiological properties of neural cells allowed them to process information. Donald Hebb was among the pioneering researchers in this field, suggesting that *learning* could result from strengthening the synapses between synchronously firing neurons. Conceptually important was also the work of Friedrich Hayek, and his hypothesis that order in the brain could arise out of decentralized networks of simple units. These ideas in turn gave rise to the first working models of neural processing. In 1943, McCulloch and Pitts published their *artificial neuron*, illustrated in figure 1. Fourteen years later, Frank

[7] An interdisciplinary branch of neuroscience involved in the neural mechanisms of cognition

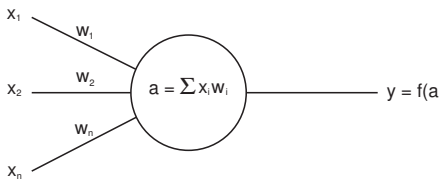

Figure 1: *McCulloch-Pitts neurons*

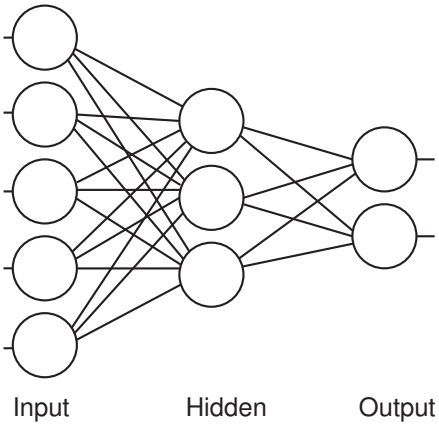

Figure 2: *Simple multilayer perceptron*

Rosenblatt developed a working neural network model consisting of McCulloch-Pitts neurons, which is known as a *perceptron*. However, after initial optimism, ANN's fell into something of an academic disrepute when Marvin Minsky and Seymour Papert proved that perceptrons are linear classifiers unable to solve classification problems that are not linearly separable, such as the simple XOR problem. Furthermore, Minsky and Papert argued that even networks with more layers would not be able to solve the XOR problem [15]. Not until the 1980's did ANN come back into favor in academia, when their conjecture was proven false.

2.3.1 Neurons - The basic building blocks

The neurons or nodes are the basic elements of neural networks. The mean number of connections feeding into a genuine biological neuron is around 10000, although the number is highly dependent on the type of neuron and where it is located in the nervous system [5]. Like their biological counterparts, artificial neurons gather input from neighboring nodes and pass it on, and can as such be regarded as functions from many dimensions to one.

In the nervous system, the effect one firing neuron has on another depends on several factors. Among them are how far from the cell body the synapse is located, the amount of neurotransmitters[8] released when the neuron fires, and the number of receptor molecules in the receiving cell. The two last factors can be modulated through learning. In the same way, the effect of one artificial neuron on another depends on the *weight* of the connection between them, as in biological networks. The activation passed on from the first neuron is simply its activity multiplied by

[8] Chemical transmitting information across the synaptic gap separating two neurons.

the weight. The activation in the receiving neuron is a nonlinear function of the sum of all incoming activation. In perceptrons, the activation of a neuron y is given by

$$a(y) = K\big(\langle w, x\rangle + b\big) \tag{1}$$

where x is a vector of activation of the neurons feeding into y, w is a vector of their associated weights, b is a constant (bias), and K is a (usually nonlinear) function such as the logistic or tanh. The neurons are individually simple, and basically only relay information coming from connected neurons. It is the nonlinear response-function of the individual neurons that makes the behavior of the network as a whole complex and difficult to analyze. The non-linearity of the response function ensures that the activation of the neuron will never exceed a maximum value, in turn ensuring the stability of the system as a whole. If the information that is fed into the system is corrupt in some way, or if an individual neurons stop working as normal, the impact is minimized. This *fault tolerance* is one of the most attractive aspects of both biological and artificial neural systems. The response function we selected for our implementation was perhaps the most classic, the logistic or sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

2.3.2 Network topology

The network *topology* refers to the layout of the network, essentially which nodes are connected, and in which direction information flows along these connections. The topology of the network is one of the most important parameters determining to what degree it is able to solve a given problem. It is common to divide the network into *levels or layers*, where nodes in one level are mostly or exclusively connected to nodes in the neighboring levels. The level that receives the vector to be processed is referred to as the *input layer*, and information propagates the network terminating at the *output layer*. Between these two, there are usually one or more *hidden* layers, referred to as such because their activity is not generally visible. It is common that information flows only in one direction through the network, which is referred to as the network being *feed forward*.

It is clear that an almost infinite number of topologies is available to a researcher for any given problem. How many layers should be used? What exact interconnec-

tivity should there be between layers, and should connections span more than one? Should we use recurrent connections, connections where information can flow away from the output layer? Fortunately, neural networks are fairly robust in that many topologies can learn the required mappings in a given task. However, the different topologies may vary in terms of speed of training, and generalization from training to test data. Though the possibilities are endless, very few guidelines exist to help select topologies.

So let us look briefly on which topological parameters can be important to the success of ANN. **General number of connections;** If topology is too small (in terms of units and connections), the network may not be able to learn the desired input/output mapping. On the other hand, if it is too large, the network very often generalizes poorly to new input [4]. All the weights need to be set by the training data, and more weights translates into a larger required training set. **Effect of the number of hidden neurons;** Usually the network will have fewer nodes in the hidden layer than in the input layer. This forces the network to extract features from the input vector, and the final decision (output layer) is based on a small number of variables [40]. As with the general level of connectivity, a larger number of hidden nodes will allow the network to adapt better to the training data. However, as with a general increase in connections, this adaptation may be "over" specific to the training data, again resulting in a poorer generalization.

Number of layers; As described above, a (two layer) perceptron is simply a linear classifier, meaning it can only solve a classification problem when a linear function can separate the classes in feature space. For many problems of any complexity, this is not enough and we must resort to more layers. Also in terms of hidden layers one should limit oneself to as few as possible, as generalizability and training speed will be affected. However, occasionally more than two layers may be the best solution. This is typically the case when very similar input vectors must result in very different output responses. With four layers, the first hidden layer can be used to recode the data before passing it on to the second hidden layer, which performs the required mapping onto the output nodes. **Full vs local connectivity;** In a *fully connected* network, each node in a layer is connected to every one in the layer above it, and this is often the default choice. One popular option is to start with a fully connected network, and *prune* away connections by testing to see if network performance deteriorates if the weights are removed. Empirical evaluations have determined that local connectivity performs better at face detection tasks [9]. **Recurrent connections;** Recurrent connections are of course the norm in biological

neural networks, but is in practice rarely used in ANNs. Partly because only certain classes of tasks truly require these connections, and also because algorithms training these networks are far less established. The weights of the connections distributed through the network can be viewed as its long term memory, adjusted to learn a given task from a training set. However, typically the network has no *short term memory*, once the information has propagated the network it cannot remember the pattern it was last exposed to or the order in which two patterns were presented. If the network is to have such a short term memory, it must have recurrent connections.

2.3.3 Network training

By network training we usually mean an algorithm for adjusting the weights of the connections in order to minimize the output error relative to some criterion. In *supervised* training, the network is presented with a training vector for the input layer, as well as a vector containing the correct responses for each training vector. The total error given a set of weights is usually formalized as the sum of squared deviations of the output nodes from the correct response.

$$E = \frac{1}{2}\sum_{o}(t_o - y_o)^2 \tag{3}$$

The development of the *back propagation* algorithm for training multilayer perceptrons was a major reason for the revival of interest in ANNs. Although first proposed by Paul Werbos in the 1970', it was after the classical article in 1986, Rummelhart et. al. in Nature, that the back propagation algorithm became widely popular [24]. Effectively, backpropagation is a way to train multilayer perceptrons using *gradient descent*. The algorithms derives its name from the fact that the error propagates the network from the output layer to the input layer, before the weights are changed. Gradient descent is an optimization algorithm that approaches a local minimum by adjusting the weights so that we move in the steepest downward direction along the error in small steps.

$$\Delta w_{ij} = -\epsilon[\frac{\delta E}{\delta w_{ij}}] \tag{4}$$

The size of the steps to take is referred to as the 'learning rate δ. Ideally we want to end up with network weights that leaves us at the lowest point on the error surface. A high learning rate will usually mean the training converges faster, but if the error surface is very eccentric, taking large steps could ignore important local gradient

information, and lead us off in the wrong direction. As the algorithm uncritically follows the gradient, it is very likely to get stuck in a poor minimum, and to reduce the risk of this, a *momentum* is usually added to the movement along the error surface. As the name implies, this parameter is intended to give the algorithm enough momentum to get out of small dips in the error surface.

In order to have a starting position on the error surface, the network weights are initially seeded with small random values. From there, training proceeds iteratively, either until the algorithm converges{we are at a minimum, and there is no further improvement, the error drops below a required value, or a maximum number of training iterations have been run. A host of other trainig algorithms have been developed, such as the *rprop* and*quickprop*, but backpropagation remains the workhorse of neural network training.

The error surface is usually very eccentric, and in genuine networks of any size, we never end up at the global minimum. However, this is not necessarily such a bad thing. The network may still solve the problem admiraly and generalize well. Furthermore, because they usually never end up with the same final weights even when trained with the same data, different networks with the same topology respond differently to new input. This can be exploited by combining the response of several networks, usually referred to as *network voting*. In other words, even if one of them makes an error, the others can compensate.

2.4 Genetic algorithms

A *algorithm* can be defined as explicit step-by-step procedure for producing a solution to a given problem. According to this definition, and the way it is commonly used in computer science, algorithms should *guarantee* that the desired end state be achieved if the procedure is followed. Different algorithms can vary in their efficiency in solving a class of problems, and can be compared by how the computer resources required by the algorithm, such as the number of instructions or temporary storage space, scales as the size of the input increases. This is formalized in *O-notation*

$$\mathrm{f(x)} = O\big(u(x)\big) \text{ as } \mathrm{x} \to \mathrm{a}, \text{ provided that } |f(x)| \leq K|u(x)|$$

However, certain classes of algorithms such as probabilistic algorithms, may not strictly fall under the definition of algorithms given above. In a probabilistic algorithm, the probability that the result will deviate from the desired end state when the number of iterations increases can be quantified, and will often converge to 0 as the number of iterations increases to infinity.

Similarly, genetic algorithms (GA) do not guarantee a that certain end-state is achieved, nor that the distance from such a state can be quantified. Instead, genetic algorithms represent a general way of moving towards better solutions in certain complex optimization scenarios. As the name implies, it borrows stylized elements and operators from evolutionary biology, and in essence, a simulated micro evolution is run on the problem domain.

For most real world problems of any size, a complete evaluation of the search space with any significant degree of resolution is prohibitive, and often there are few reasoned choices as to how to move towards an optimal solution. Strictly optimal solutions, as the global minimum on some error function, may not even be necessary. What may be desirable is a non-arbitrarily improved solution.

Genetic algorithms may be particularly well suited for complex problems where heuristic or more formally defined methods of optimization are not available, where the fitness function is so complex that traditional methods frequently end up at local minima, the search space is large, complex or poorly understood, or domain knowledge is scarce or expert knowledge is difficult to encode [4].

Before a genetic algorithm can be implemented for a given problem, two elements must be in place. These are i) a way of representing a solution in such a way that it can be manipulated by the algorithm, and ii) a way of determining the fitness of a given solution.

2.4.1 Elements and operators in genetic algorithms

Genes; representing solutions. In GA a solution to a problem can be thought of as an individual in a population of solutions. The solutions to the problem must be coded in a way that can be acted upon by the evolutionary operators. As this is roughly analogous to the way DNA contains the biological blueprint of an individual organism, the encoding of a solution is often referred to as a chromosome. In DNA, triplets of 4 different kinds of bases code for 21 distinct amino-acids. Though computational analogues of such complex encoding schemes are possible and have been implemented, usually a simpler encoding scheme is used, such as a bit string. The important point is that this chromosome codes for one unique solution.

Fitness function; evaluating solutions. As evolution is based on the "survival of the fittest", we need some way of determining how fit our solutions are, and hopefully move toward better ones during evolution. More specifically we need a *fitness function*. This function takes a chromosome, decodes it into a "working

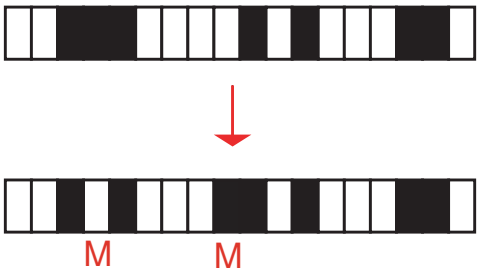

Figure 3: *Bit mutation*

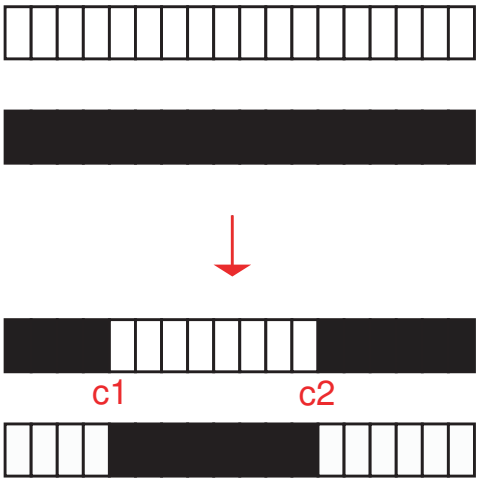

Figure 4: *Chromosomal crossover*

individuals", and evaluates it's fitness. What constitutes the fitness of a solution must be defined for each problem or class of problems, and how good the fitness function is will largely determine the success of an evolutionary approach. Unless the fitness function is handled properly, GAs have a tendency to converge towards a local rather than the global optimum of the problem.

Genetic operators; modifying solutions. Given than we now have ways of encoding individual solutions, and can determine their fitness, what we need are evolutionary operators acting upon the individuals. Two main operators are used, *mutation* and *crossover*, and their relative importance is widely debated. Both are illustrated in figure 2.4.1.

Mutation is perhaps the most widely known evolutionary operator. It basically refers to small random alterations to the chromosomes. An important parameter of evolutionary algorithms is the rate of mutation, the balance between change and stability. If the rate is too low this may cause "genetic drift", or convergence to local optima. Alternatively, if the mutation rate is too high, it may lead to unstable chromosomal behavior and "jumping around" in search space. However, exactly what constitutes a good mutation rate is hardly ever discussed, and for all practical purposes must be chosen based on trial runs.

The second evolutionary operator is that of crossover. The cells that fuse in conception each have a single version of the 23 unique chromosomes in the human genome. This fusion results in a diploid organism, one with two sets of each chromosome. During meiosis[9], when the sex-cells are generated, the pairs of chromosomes separate, one migrating to each of the two new cells. However, in the process, the

[9] A cell division that generates haploid sex-cells, i.e. cells with only one set of chromosomes.

strands of the two versions of the same chromosomes can *cross over*, resulting in realignment of parts of the DNA threads. A similar process is often implemented in genetic algorithms. When two organisms are selected for breeding, there may be one or more points of crossover, where large segments of their DNA is exchanged. Crossover thus works on a much higher level than mutation, exchanging large chunks of DNA, and therefore also larger number of phenotypic traits.

Up to now, a fairly stylized but biologically reasonable set of operators have been implemented. However, as the rates of mutation and crossover are hard to calibrate, and the gene pool tiny compared to that in say, a small drop of water, you risk loosing good solutions from the pool. To ensure that good solutions are not lost, a process referred to as elitist selection or *elitism* is used. This merely refers to bypassing the regular evolutionary processes, and simply copying a handful of the best solutions in one generation into the next. This way elitism can rapidly increase the performance of genetic algorithms, as it prevents the loss of good solutions.

2.4.2 Running a genetic algorithm

We are now in a position to put all the elements together into a working genetic algorithm. First, a pool of (often random) chromosomes is initialized. A random pool ensures that initial solutions are distributed over much of the parameter space. To ensure that large parts of the parameter space are searched, the pool should ideally consist of a large number of chromosomes, from hundreds to thousands. The exact number is of course up to the individual researcher to determine, and will depend heavily on the time taken to evaluate fitness. For each generation, the chromosomes undergo some degree of mutation. Pairs of chromosomes are then selected for breeding, the probability of selection being some function of their fitness. Mating results in a new pool of chromosomes, with hopefully better mean fitness, and the process can repeat itself. The algorithm can terminate after a fixed number of generations or when a distance from a criteria is reached, while no convergence to such a minimum distance is guaranteed.

2.4.3 Genetic algorithms for neural networks

With the number of synapses in the human brain estimated at 10E14 [37], and the number of genes in our genome currently thought to lie between 20-30 000, it is unlikely that evolultion has shaped the topology of this network at the level of individual synapses. However, on a slightly higher scale, the brain is the result of

evolution, and if artificial neural networks really mimic characteristics of genuine neural circuits, it is not far fetched to try to shape them with the same kinds of evolutionary processes. In fact, applying evollutionary algorithms to various characteristics of ANNs is not new, but has emerged as an established method of both *selecting network topology* and *training networks*[4]. Until recently, selecting network topology was done trough human expert knowledge, but this may change. Schaffer et.al. have presented results showing that the ANN designed by evolutionary approach had better generalization ability than one trained by backpropagation on a human designed architecture [4]. Furthermore, while the backpropagation algorithm is by far the most widely used method of training network, it may not always be an option. Training through gradient descent approaches requires the error be differentiable, while evolutionary algorithms can train networks regardless of whether they are feedforward or recurrent, and even if error is discontinuous [4]. Evolutioinary training is also less likely than graidient descent based methods of getting stuck in a local minimum, because it searches over several regions in parallel. Unfortunately, for the same reason it has dificulties in fine tuning the parameters.

3 Face localization

3.1 An overview of the algorithms

Though face detection has received a lot of attention the past 10 years, it has been a field of research for many more. With the development of better algorithms and more powerful computer hardware, the field has moved from searching for a single vertical frontal face in visually simple images with homogeneous background, to multiple faces in difficult lighting conditions, at various angles and distances from the camera, situated in complex visual scenes [19]. Still, even today the majority of algorithms published are only able to detect vertical or nearly vertical faces.

As outlined in the introduction of this thesis, the ability to quickly and robustly localize faces in images is valuable in numerous areas, ranging to surveillance, content based indexing and human machine interaction. For this reason there are a multitude of published algorithms in this area, each one trying to claim superiority on robustness or speed.

In the myriad of approaches, it is useful to distinguish two major categories [39][19]. One assumes an a priori face model (model-based/top down), and attempts to localize/validate by means of an explicit interrelation of facial features (such as eyes, mouth and nose), while the other approach assumes no prior model.

3.1.1 Feature-based/knowledge-based algorithms

Basically, in feature-based methods for face localization, we attempt to use what we already know about faces to construct a model which can be compared to the data. When presented with a new picture, a feature based algorithm usually begins with a low level analysis, where properties such as edges, colour, motion and texture are extracted [19]. The constellation of these features are then compared to the model. *Deformable templates*, templates that fit an a priori elastic model to facial features are an example of a more sophisticated feature based approach. A problem with several of these methods, is that the deformable templates must be initialized in the vicinity of the face. Good face candidates can be found by looking at elliptic skin-colour regions.

3.1.2 Image based algorithms

The defining characterizing of image based approaches is that we do not directly encode our preexisting knowledge about facial features into the algorithm. Rather,

Figure 5: *Input image*

Figure 6: *Thresholded*

Figure 7: *Clustured*

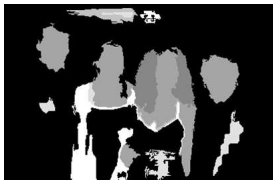

Figure 8: *Cleaned*

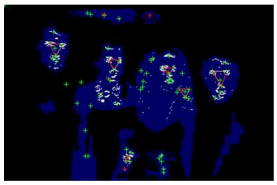

Figure 9: *Highpass-filtered*

the system learns to discriminate faces from non-faces through analyzing a training set. The images are is mathematically compared with the stored representation without explicitly extracting facial features. Popular methods here include principal component based algorithms, neural networks, support vector machines and hidden markov models [19]. Principal component based methods are presented in depth in section 4.1.2, while a neural network based approach is descibed in 3.4.

3.2 Illustrating challenges with a model-based algorithm

Implementing the neural network based algorithm presented in this thesis was a laborious process, undertaken only after an initial model-based approach proved not to be sufficiently robust. Model-based approaches are often fast to implement and execute, are easy to understand, and have an intuitive appeal. They do however rely on the assumed faces conforming fairly strictly to the model, and small deviations from ideal conditions will often result in failure. In this section I will sketch the model-based face localization algorithm we abandoned, and give examples of some of the many ways it can fail. We hope this will illustrate some of the difficulties involved in automatic face localization. The model-based attempt is loosely based on the algorithm of Hesieh et. al.[12]. The steps of this algorithm are, *skin colour thresholding, skin colour clustering, feature extraction, template matching*, and these

steps are illustrated in figure 5.

1. Skin-colour thresholding.

The simplest condition for face localization algorithms is a single frontal face against a distinctly non skin-coloured background, such as a typical passport photo. Under these circumstances, skin colour thresholding can be almost sufficient to localize the face. If an area is skin coloured, has roughly oval shape and contains some darker internal regions, we may classify this as a face. Figure 6 shows the result of skin colour thresholding of figure 5.

2. Skin colour clustering.

Figure 10: *Clustered 2d histogram*

In natural settings, skin colour thresholding will hardly ever be sufficient for localization because variations in background hues will make faces difficult to segment based on colour alone. Often two or more faces will be merged in a single skin coloured region, as can be seen in figure 6. To determine whether an image area contains several faces, some sort of *clustering* is typically performed. In our initial approach, we developed a clustering algorithm for skin colour regions, the details of which are given in appendix A. Essentially the clustering approach involved identifying the peaks of the 2D hue/saturation histogram[10]. After merging close peaks, the histogram is divided into regions according to the euclidean distance to the center of mass the closest histogram peak. Clustering of the thresholded 2D hue/saturation histogram can be seen in figure 10. Figure 7 illustrates the effect of the clustering algorithm, and figure 8 after some cleaning. We can see that in this case the clustering algorithm has successfully separated the different faces, and at most one face is present in each skin colour region. For more examples of skin colour clustering, see appendix A. While we were pleased with the performance of the clustering algorithm[11], occasionally it was unable to separate a face from the background or from another. This was typically the case if the background colour was similar to that of the face. As it only searched for a single face in each skin coloured region, one or often both were missed if the region contained more than one face.

[10] peaks were defined as >0.9 standard deviations above the mean.

[11] See website for more examples

Figure 11: *Example of successful model-based localization*

3. Feature extraction.

Hopefully at this stage, the image is broken down into a set of regions representing different objects or people, and the next stage is to determine whether any one of these represents or contains a face. This is done by first extracting low level features from the regions. Eyes, nose and mouth are often darker than the surrounding area, so a common step is to search for patches of pixels with lower intensities than their surroundings [12]. We found this approach to be too unreliable, and easily affected by shadows and normal variation in light intensity. Rather than look for darker areas, we found high-pass filtering to be more robust. However, it becomes a challenge to detect faces at various scales, because highpass filtering of larger faces will leave very different signatures than small ones. Different high-pass filters may be best suited to enhance the facial features in large vs. small faces. Different features could be used for skin-colour regions of different sizes, but the size of this region is not a perfect indicator of the size of the face within. Although better than mere intensity, the highpass filtering was also sensitive to edges caused by glasses, beards, sharp shadows etc.

4. Template matching. Once the features are extracted, one must determine whether the *constellation* of features found represented a face. Hsieh et. al. used a template of light intensity roughly representing what one would expect given a vertical face. i.e., two darker objects in the left and right upper quadrants, a long dark patch in the lower middle etc. Our approach was to fit a triangle to the center of mass of the most prominent highpass-filtered objects in the *interior*[12] of the region,

[12]Search restricted to the interior region to avoid highpass filtering the border to the background.

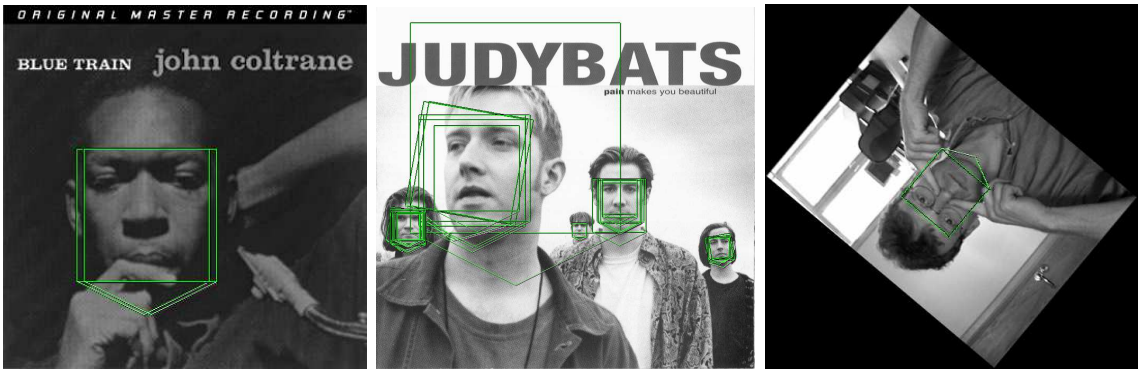

Figure 12: *Challenging images for localization algorithms. Green squares indicate positive hits from our implemented system.*

as it was assumed that the most prominent objects would represent the two eyes and the mouth. If such a triangle fit well[13], then it was declared a hit. Searching for triangles representing facial features is a widely used strategy [17]. Usually it is assumed that the faces are vertical or nearly vertical, and if this is not the case then they will be missed. Approaches that require all features to be found, will also not detect faces where one of these are occluded.

Figure 11 illustrates the success of the model-based localization algorithm on an "easy" image. We can see how the eyes and mouth are the most prominent objects after highpass filtering, and the triangle fits perfectly. On the other hand, figure 12 shows examples of images in which the model-based approach would result in detection failure. Factors that make these images difficult is the lack of colour, difficult lighting conditions, rotation and occlusion, and the vast difference in scale among the faces. One might argue that a face recognition system does not need to be robust under difficult conditions. This is because the current recognition algorithms are all so sensitive to the the same variations in lighting and occlusion that are likely to throw off a model-based localization algorithm. However, it was our experience that often the algorithm would fail even under seemingly good conditions. Furthermore, even when a face was correctly detected, this did not necessarily mean that the algorithm was able to bound the face sufficiently well for recognition. Often it confused a mouth with the shadow under the nose, or the eyes with the eyebrows or rims of the glasses, resulting in a bounding box covering either too little of the face, or too much of the background. For an example of localization failure, see appendix A.

[13]the length of all sides were roughly equal

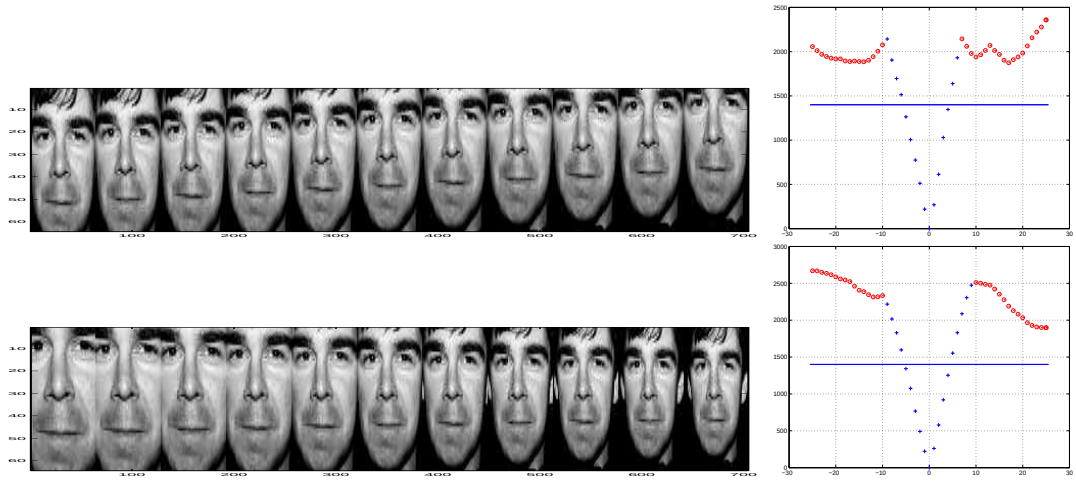

Figure 13: *Horizontal line on figure to the right indicate recognition threshold. Translation and scaling of even small amounts leads to recognition failure.*

3.3 The importance of accurate localization for recognition

Face recognition algorithms are often tested on variations in lighting, pose or facial expression relative to the stored template, in an effort to determine how robust the algorithm is. However, rarely do authors explore how sensitive these algorithms are to sub-optimal segmentation of the face area. In other words, how accurately must the coordinates of a bounding box around a face be in order to be able to recognize the individual, and how does performance degrade as a function of different kinds of sub-optimal localization. As we had decided to use the eigenface approach[14] for the recognition we first wanted to get an impression of how sensitive it was to the accuracy of the localization.

A prototype of a eigenface based recognition system was implemented in MATLAB. We then selected 31 images from the Stirling face database [29], and manually found the coordinates of the faces. Recognition rates were evaluated as a function of systematic variations in these coordinates[15]. The kinds of variations performed included horizontal, vertical and diagonal translation, rotation and scaling. Figure 13 illustrates the approach, and the graph to the left shows how change in position of the bounding box increased the distance from the stored template[16]. The prototype indicated that in the images we used, and at a scale of 64×64 pixels, a vertical or horizontal translation of more than 4 pixels usually resulted in a misclassification, as did rotation > 6 degrees. The exact numbers are not important, but our trials

[14]The details of the eigenface based approach are given in section 4.2

[15]Faces were downscaled to 64×64 pixels before recognition

[16]The maximum distance for recognition was set to 1.4E3, based on initial trials

indicate that at least the eigenface approach is sensitive to poorly segmented faces, and the bounding box does not have to be much off for the algorithm to fail.

A neural network based approach will usually indicate hits in multiple adjacent locations. Given the results above we therefore propose not to search overlapping face hits for the single best match, but rather attempt to recognize each box individually. This may increase recognition rates, as chances are that at least one of the bounding boxes sufficiently well segments the face for the individual to be classified correctly. On the other hand, false positives or poorly segmented faces will not be recognized, and will be discarded during recognition.

3.4 Rowley's neural network-based approach

Although a multitude of neural network based approaches to face localization have been published, Rowley, Baluja and Kanade are usually credited with the pioneering algorithm. A neural network-based upright frontal face detection system was published in 1998 [22], and later the same year it was extended to handle faces rotated in the image plane [23]. Subsequent approaches usually borrow one or more fundamental elements of their algorithm, the major ones being an image pyramid, a router network and a detection network.

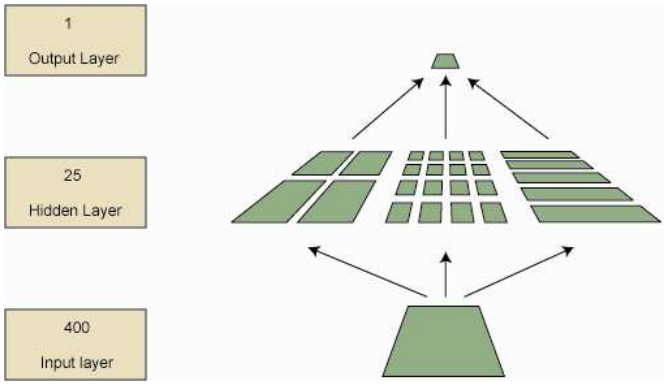

Figure 14: *Topology of Rowley's face detection network*

The neural networks receive their input from a 20×20 sliding window centered around every pixel in the image. A 20×20 sliding window has been popular also in vector machine and likelihood-based approaches to face localization [25]. As this window is of fixed size, the only way to detect faces at different scales is to scan the image at increasingly lower resolutions. Therefore, given an input image, an *image pyramid* is first constructed. In Rowley's implementation, this pyramid is construct by sub-sampling the original image. At every location, the sliding window is histogram equalized before it feeds into the *router network*, which is trained to estimate the amount of clockwise rotation of an area. The histogram equalization is performed to compensates for differences in camera input gains, and improves

contrast.

Theoretically, rather than using a router network, each location could be subjected to 35 detection networks, each trained on a specific face orientation. However, the neural networks are fairly computationally demanding, and this approach would require a multiple of 35 scans compared to the router approach. For the router network, Rowley uses a fully connected three-layer perceptron, with 400 input nodes, 15 hidden nodes, and 35 output nodes. The network is trained on images of n$\times$10 degrees rotation. The most active output node represents the estimated amount of clockwise rotation. The window can then be rotated anticlockwise the estimated number of degrees. A bivariate linear regression is used to reduce the impact of directional lighting. Following the lighting correction, the window is histogram equalized once again, and presented to the *detection network*. As the name implies the task of the detector network is to classify a given window as a face or non-face. The topology of the detector network is sketched in figure 14.

It is clear that this approach is very computationally intensive. A 800x600 pixel image will give rise to more than 750 000 20×20 pixel windows to scan. Each window must be equalized, rotation estimated, rotated back, fitted to a bivariate linear regression, equalized again and run through several detector networks. Rowley's router network consists of 6530 network edges, so a single rotation estimation will quickly involve more than 10000 floating point operations. As well as being computationally intensive, the vast amounts of windows scanned, usually only a few contain a face. The large number of trials and the low base rate can quickly translate into a lot of false positives. To counteract this, Rowley uses the following strategies; an *arbitration scheme*, *bootstrap training* and *network voting*.

Their arbitration relies on simple heuristics, such as disallowing overlapping faces, and requiring multiple hits in the same image region. Bootstrapping in this context refers to running the algorithm on images without faces, and adding the false positives to the training set. Another method of reducing the number of false positives is using several detection networks, each trained with different initial random seeds to validate the presence of a face. How the decisions of these individual classifiers in turn are treated has been the subject of some investigation in the literature (see ex. Hjelmås [11]). The network voting involves AND'ing the response of two networks trained on different initial random values.

3.5 Our face localization approach

Our approach uses the main constituent part of Rowley's algorithm, while we have attempted to reduce running time. This is primarily done through:
i) using a *skin colour* and *image texture* model to substantially reduce the search space.
ii) Using an evolutionary strategy to reduce the number of edges in the router network.

3.5.1 Preprocessing: Colour transformation and constructing the image pyramid

Our program accepts uncompressed TIFF [1] images as input. When the program is run, the image is first checked for colour information. If the input image is in RGB format, it is transformed into an HSI colour space. The details of this colour transformations are given in section 2.2. The resulting image has three bands. One consisting of hue (H), one of saturation (S) and the last of pixel intensities (I). In our implementation, the colour information is solely used to discard non-skin-colour regions. The detection step itself is colour blind, and operates exclusively on the I band of the image. Where as Rowley used a sub-sampling approach in constructing the image pyramid, we eventually settled on bilinear interpolation in downscaling the image. A sub-sampling approach resulted in large deterioration on higher levels in the pyramid and poorer detector performance, while the extra processing required for the interpolation is minuscule compared to the other computationally intensive steps in the algorithm.

3.5.2 Optimizing the search by using colour and texture

The main limitation of a neural network based approach is that it is very computationally demanding. Rowley's 1998 implementation used 383 seconds to scan a 320×240 pixel image for vertical faces[17]. With the more densely connected router network, and additional operations of rotation and equalization, the time for detecting rotated faces will easily be a multiple of that for vertical. In fact, it will be shown in section 6 that compensating for face rotation increases processing time in our implementation by up to 500%. If the algorithm is to be genuinely useful in traversing large databases, the running time must be reduced substantially. The

[17]Benchmarks performed on a 200MHz R4400 SGI Indigo 2 system

most obvious way of optimizing the algorithm is searching fewer windows, by discarding areas of the image surface where the presence of a face is highly unlikely. With almost all running time of the algorithm tied up in the neural networks, discarding a percentage of the image results in an almost linear reduction in running time. As can be seen in figure 3.5.2, often substantial regions of the original image can be ignored if the image is thresholded on texture and colour. While several authors have suggested using skin colour to reduce the search space, we have not seen a similar argument presented for texture, although numerous papers present texture based methods for localizing faces [19].

Empirical studies have demonstrated that skin colour of people of different ethnicity vary mostly in light intensity, and relatively little in chromatic quality (under normal lighting conditions). Skin colours were defined as an area of the HS plane, with hue values ranging between <0,130>, saturation values between <0,70>, (and an intensity level >40).

Colour can be used in several ways, either by discarding non-skin colour areas, or by lowering the detection threshold in these areas. In our approach, we simply discard those areas of the image that fall outside the thresholds.

If an image contains a single large face, it will still be scanned on many different resolutions, most of which are many levels below the one on which it is detected. In fact, by far the most windows will be scanned in the lowest levels of the pyramid. At these bottom levels 20×20 windows centered over an area of the face will contain skin-coloured pixels, but have a texture that lets us discard it. Furthermore, in grayscale images, which is still the norm in most available face databases, we obviously cannot use skin colour. At the 20×20 pixel levels at which faces are detected, they have textures that set them apart from most non-faces, so prior to detection the image can be thresholded on texture. In selecting textures to calculate, we must of course choose ones that are able to discriminate well between faces and non-faces. However, as up to a million windows must be evaluated for each image, good texture candidates must also be fast to calculate. Based empirical evaluations we selected *variance* and *entropy*.

Variance is here defined as:

$$\frac{1}{n-1}\sum_{i=1}^{25}\sum_{i=1}^{25}(X_i - \bar{X})^2 \tag{5}$$

This property is used because under normal lighting conditions faces have areas with lower light intensities (eyes, mouth, shadow under nose etc.), and better lit

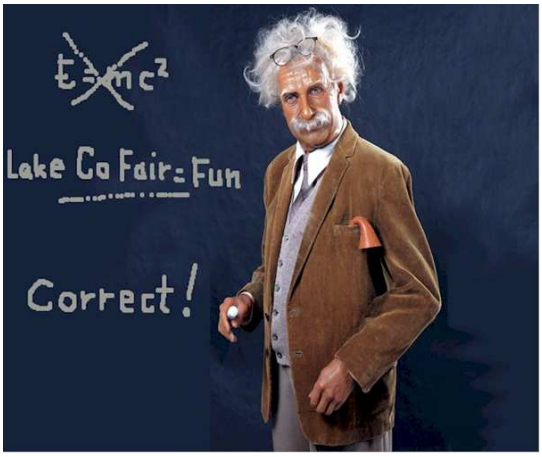

Figure 15: *Input image*

Figure 16: *Texture mask*

Figure 17: *Colour mask*

Figure 18: *Combined mask*

areas (tip of nose, forehead etc.). In contrast, homogeneous backgrounds have low variance.

The second texture property we used was entropy, defined here as:

$$\sum_{i=0}^{2^b-1} p_i \times log_2(p_i) \tag{6}$$

where p is the normalized histogram of the cell. Entropy quantifies the amount of information needed to code the histogram. As computing entropy requires more floating-point operations than variance, it is only calculated in the regions that have demonstrated sufficiently high variance.

In order not to compute the texture and colour properties in every scanned window, we set up a masking map in every level of the pyramid. Each cell in the masking map corresponds to a 25×25 pixel region of the given level. If the region does not contain a certain amount of skin colour or the necessary texture properties, the index in the masking map is marked with 0, indicating that the 25 windows in this region are to be skipped. In the final version of the algorithm, variance threshold is set to >500, and entropy to >3.6.

In addition to reducing the running time of the algorithm, the colour/texture thresholding substantially reduces the number of false positives. This is because windows taken from homogeneous image regions can give rise to face-like patterns once these windows have been histogram equalized.

3.5.3 Training the networks

It is the process of network training that makes the ANN approach so flexible. We do not need to present the distinguishing features of faces to the classifier, but if the training converges, such distinguishing properties are found by the network itself. Care has to be taken to ensure that the features are substantive in discriminating the classes, and not just an arbitrary difference between the training set and the test set[18]. If the latter is the case, the network will generalize poorly to future data.

During training, the network must learn to distinguish faces from non-faces. As explained previously, this is done by modifying the weights when the network responds incorrectly. For the network to learn to discriminate the classes "face" and "non-face", the training set must contain a large number of representative samples

[18]If the bounding box is set too large, the colour of the background can become such an arbitrary feature

Figure 19: *The making of training vector for router network*

from each of these classes. Although it is fairly straight forward to select faces which contain a fair amount of the within-class variance one can expect, such as the configuration of facial features (distances between eyes, size of nose etc.) facial expression (smiling, angry etc), occlusion and so on, it is much harder to captures the variance in the class "non-faces". To construct the training vector, 192 faces were selected from the Stirling face database [29], and the Yale face database [38]. Two separate databases were selected as their background on the images are in turn black and white. Because there will often be some background present in the segmented faces, we must ensure that the network does not rely on background colour in detecting faces.

The coordinates of a square beginning slightly above the eyebrows and down to below the mouth were found manually. MATLAB was used to rescale these faces to 20×20 grayscale images. To increase the sample size, versions of the original face that were mirrored and also translated a pixel in each direction were added to the training set. Non-faces were found by sampling 20×20 pixel windows from image pyramids of satellite photos.

Making training data is a fine balance. This system is ultimately meant for face recognition. False negatives are more damaging than false positives, especially as positives most likely will be discarded in the recognition step. A bootstrap procedure was therefore used, but the number of false positives added to the training set was restricted to 1500. When a multiple of this was used, the number of false negatives dramatically increased.

The final training vector for the detector network contained 1214 faces, 11808 non-faces, and 1500 bootstrapped non-faces. The network trained with momentum α=0.5 until the output error was less than 0.02. At every iteration, the sequence of training stimuli was randomly scrambled.

To train the router network, the coordinates of the bounding box of the vertical faces were rotated 35×10 degrees, and a separate image sampled at each point (see fig 19). We selected the sigmoid activation function for the neurons. Because of this, we cannot use the raw pixel values as input to the networks, as the sigmoid function

Figure 20: *EA fitness over generations*

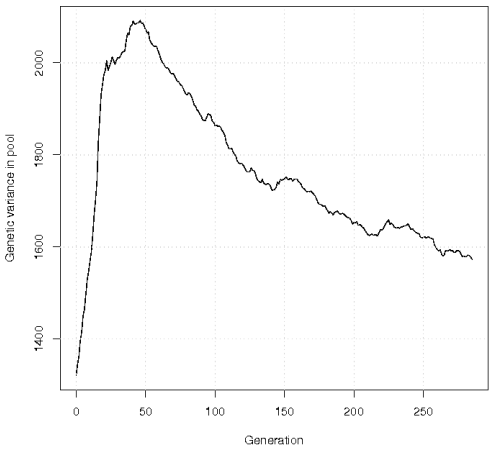

Figure 21: *Variance in gene pool*

approaches the vertical asymptotes at x-values larger than 4. Instead, pixel values were transformed from [0,255] to [-6,6] so as to best cover the range of the sigmoid function[19].

3.5.4 Using an evolutionary algorithm to improve the network topology

Rowley's router network is fully connected, meaning that a node in a given layer is connected to every node in the two neighboring layers. Fully connected networks are a popular default choice, not necessarily because they are best suited for the task, but because alternative topologies based on a theoretical or substantive rationale or documented performance are not available. Neural networks are highly nonlinear systems, and their behavior is hard to predict. Some classes of networks, such as those with feedback loops or more than three layers can be recommended for certain tasks, but there is little rationale for preferring one specific topology within these classes. This is not to say that the specific topology is unimportant, just that we rarely have few genuine theoretical reasons to choose one over another, which often leaves us with the fully connected default.

The fully connected topology has at least two drawbacks. Firstly, the larger number of connections a network has, the better it will be able to adjust it's responses to a training set. This will in turn often make it "over-trained", meaning that it is poorer at generalizing correct responses from the training set to new instances. Secondly, more connections translates into more mathematical operations. Reducing

[19]transformed value = [(pixel value)/21.25]-6.0.

the number of connections will in theory have a nearly linear effect on the running time, although the effect will in most cases be even (and sometimes substantially) larger, given that large networks take up much memory and cache may become a bottle-neck.

What we need is a theoretically justified approach to selecting an optimized topology, and one established method for doing this is by using evolutionary algorithms (EA). To our knowledge, the only published article using evolutionary algorithms for face detection is that of Wiegand et al [35].

There is an interesting parallel between evolutionary algorithms and neural networks. We are not enforcing a specific solution based on prior knowledge of the specific domain, but rather implementing a general algorithm which itself finds a way to solve the problem. In fact, we might never fully understand what information the system uses to discriminate between classes.

Training a single network can take a long time. To make an evolutionary approach at all feasible, a large number of generations must be run. Also, the gene pool must contain enough "chromosomes" and these must represent enough genetic variation if the algorithm is to avoid getting stuck in local minima. This translates into thousands of individual networks trained and tested. Each session must be brief enough for this to be possible.

A bit chromosome was defined so as to code for a specific network topology. The relation between the chromosome and the network is fairly simple, and a set bit represents a connection between node X and Y, while a zero bit represents the absence of a connection. In evolving our router network, we initialized the gene pool with 140 chromosome coding for networks with random connectivity.[20]

For each generation, working neural networks are made from each chromosome. These are trained for 145 iterations, which is sufficient to observe the error rate begin to flatten out, while few enough to feasibly run a greater number of evolutionary generations. During evolution the training set is fairly small, containing 30 faces rotated n*10 degrees, resulting in a training vector of length 1080. The training vector is kept relatively short in order to reduce the time of each training iteration. The important question however, is how well the network generalizes to new instances. As the most computationally demanding aspect of the evolutionary procedure is the training, the testing vector can afford to be comparatively longer,

[20]The algorithm was also run with fully connected networks in the initial pool, but these take far longer time to train, and the performance of the randomly connected networks converged to the level of the fully connected ones fairly quickly.

and in our implementation, we used 334 faces, again rotated 35*10 degrees, resulting in a vector of length 12024.

The purpose of an evolutionary algorithm is to evolve a solution that is progressively better by some selected measure of fitness. It is therefore critical that the *fitness function* is sensitive to and representative of improved performance. A poorly selected fit function renders the evolutionary approach meaningless. As with network training, the fitness of a given router network was judged as 1 - the mean squared distance of the output vector from the correct output vector. However, the correct output vector requires 35 of the output nodes to zero, and only one to be 1. It would therefore be very easy for the network to evolve to a solution where it simply scored zero on all output nodes. For this reason the error on the output node representing the true amount of clockwise rotation, and that should approximate 1.0, was weighted 10 times that of the other output nodes.

A superior network topology is a tricky thing to evolve. The vast amount of similar networks can achieve very similar levels of error on a given test set, and if the topological difference is small, it is quickly overshadowed by the effect of different initial values on the weights. In a network as large as this one, it will be impossible to discriminate the effect of removing or adding a single connection, from the effect of starting with a different initial random seed. We could increase the mutation rate on the pool, but this could potentially result in the algorithm jumping too much around in search space. For this reason we chose each mutation at the genotypic level to represent a relatively larger change on the phenotypic. On the input-hidden layer, a single change in a bit value will set the 3×3 window surrounding it to the same state as itself. This represents a more dramatic change of the receptive field of the neurons in the hidden layer, one that is more easily visible through the noise of the initial random values. Mutation in the hidden layer only affects the single weight in question. This is because there are comparatively fewer connections to alter, they encode more higher order features, and a change in any has a more dramatic effect on the performance of the network. This way a mutation became a somewhat more significant event, and its effect on the test set will be discernible through the effect of random initial weights. The probability of mutation of each bit was set to 0.0012%, and the probability of a crossover during mating was set at 0.0005 % at each location in the chromosome.

140 chromosomes is far less that what I would have preferred, but we ended up with this number as a result of the tradeoff between pool-size and generations. With such a small number of chromosomes, it becomes all the more important that there

is genetic variance in the pool. As in real gene pools, little variance in genetic data means the pool is far more limited in the solutions if can come up with. Figure 21 shows a plot of the genetic variance in the pool as the evolution progresses, while figure 20 shows improvement in fitness during evolution.

Algorithm 1 Evolve network topology

```
pool = initialize 140 chromosomes coding for random connectivity.
while not max generations do
  Mutate chromosomes in pool ()
  for all chromosomes in pool do
    net = chromosome to network ()
    train network (net)
    test network (net)
  end for
  sort (pool)
  weight chromosomes by nonlinear rank-order (pool)
  copy 4 best directly to new pool
  randomly select two chromosomes weighted by location in rank
  for each selected pair do
    apply crossover
    copy result into new pool
  end for
end while
```

Perhaps the most promising area of the algorithm to apply an evolutionary approach is to the router network. A fully connected network may be reasonable as there are few heuristic / biological or other reasons to endorse a specific architecture, but it has the disadvantages listed earlier. Starting from a fully connected router, reducing connectivity may improve performance as well as result in fewer connections, and hence faster execution.

The rotation network was evolved over 43012 minutes (30 days) on an 64bit Athlon 3000 running Gentoo linux 2.6.11-r1. After evolution was terminated, all chromosomes in the pool were evaluated, the best one was selected and trained for 1000 iterations on the full dataset.

3.5.5 Reducing false positives

The *sensitivity* of a classifier refers to it's ability to identify the presence of a target, while *specificity* refers to it's ability to reject non-targets. In practice there is usually some trade off between these two qualities, and we will never achieve perfect specificity. This is a problem in our approach, because of the high number of windows

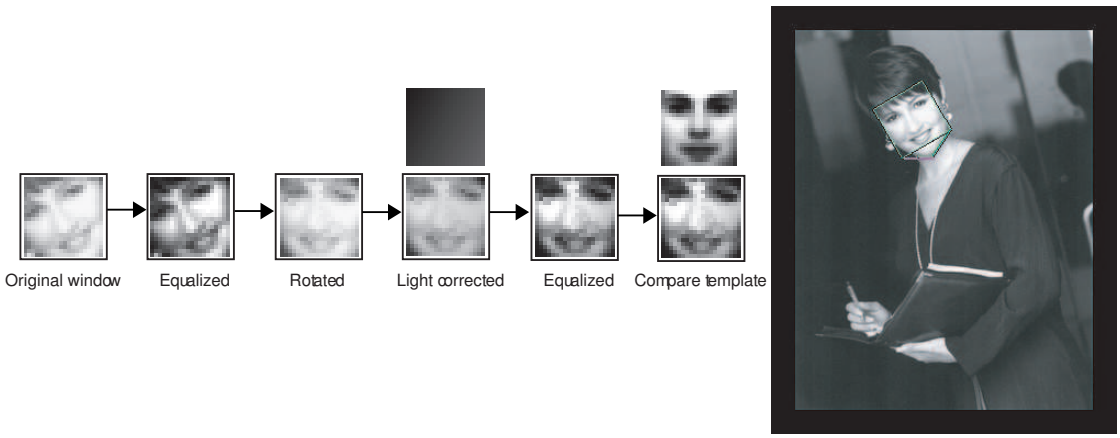

Figure 22: *Steps in the localization algorithm*

scanned, and the extreme difference in prevalence of these categories. When up to a million windows are independently evaluated, of which only a handful represent a face, even very high levels of specificity can give rise to many false positives. The router network makes the problem even worse, as all windows are rotated so that they maximally resemble a face before presented to the detector networks.

To reduce the number of false positives, we have taken the following steps. We have adopted Rowley's *bootstrap* procedure, and included 1500 vectors in the training set that were incorrectly classified as faces. We also made use of *network voting*, where the combined activation of three networks trained with different initial weights must be over a threshold value.

A further way to reduce false positives is to compare it with the surrounding areas. As the network is trained with a lot of different faces, it will typically respond strongly even to small changes to these, such as a slight translation or rotation by a few degrees. False responses are less robust to such changes. Once a potential face has been detected, the surrounding region is scanned at the same orientation as the potential hit. A potential hit is dismissed if the combined local response is below a threshold[21]. Lastly, we used a coarse template match, by requiring a hit to be less than a certain distance from the mean face vector from the training set[22].

3.5.6 Putting it all together: Localizing a face

The localization algorithm is explained in detail in figure 22. The program

[21] Threshold for the combined response in the three networks in the 8 neighboring cells was set to 15.0.

[22] $\| W - F_{mean} \| < 2000$, where W is a 20×20 window, and F_mean is the mean face from the training set

Algorithm 2 Full face localization algorithm

```
IMG ← Load TIFF image
if IMG is a colour image then
  Transform image to HSI colour space
end if
Make image pyramid
Make masking pyramid
for Every level in pyramid do
  for Every pixel in IMG do
    if Region masked then
      Continue to next pixel block
    end if
    center router network on pixel
    Equalize window
    Run window through routing network
  end for
  Rotate window anticlockwise estimated degrees
  Equalize window
  Fit linear light model
  if Distance from mean face>2000 then
    continue to next location
  end if
  Run window through 3 detection networks
  if Combined network response>2.95 then
    Check 8 neighboring cells at same orientation
    if Combined neighborhood response >15 then
      Mark hit on input image
    end if
  end if
end for
```

transforms an RGB colour image to HSI, and thresholds on skin colour. Based on the thresholded image and texture information, a masking map for every level in the image pyramid is made. A 20×20 scanning window begins to traverse the pyramid in 5×5 blocks, beginning at the top level of the pyramid. For every location, the window is histogram equalized and run through the router network. The estimated amount of rotation is in our implementation simply the output neuron with the highest degree of activation. Different strategies have been implemented, where the neighboring nodes contributed to the response of a node, but these were ultimately discarded. The window is then rotated back the estimated amount, histogram equalized again. A linear light model is fitted, and increase is subtracted from the original.

At this point we found it useful to perform a template match with the mean face from the training set. The comparison is very liberal, and the purpose is not to find faces, but reduce false positives and not waste time running the detection networks. This template match is implemented as a simple vector norm between a given window and the mean face, $\|W_i - \Omega\|$, where W_i is a given window and Ω is the mean face. If the distance is greater than 2000, the window is discarded. The window is then processed by three neural networks trained with different initial weights. If the response of any of these is less than 0.95, the algorithm branches. A potential hit is validated by analyzing 20×20 windows surrounding the 8 neighboring pixels at the same orientation. If the combined activation of the local area is greater than 15, a bounding box is marked on the input image.

Although we have attempted to select the best hit among the various candidates in a local area by comparing all responses at the same and different levels in the pyramid, we eventually discarded this approach and allowed overlapping face hits on lower levels. This was because recognition is the ultimate purpose of the system, and false positives could mask genuine hits. Poorly circumscribed faces will most likely not be close enough to any of the template vectors in the database to be recognized. So comparing each hit from the localization algorithm to the face database individually will increase the chances that at least one of them is sufficiently close to the stored template to be recognized.

3.5.7 Other possible optimizations

The main strategy adopted in our approach for reducing the running time of the algorithm is thresholding colour and texture. Even then the algorithm is computationally demanding and does not approach real time. However, the more is known

about the image, the more specifically we can tailor the many parameters to the algorithm to perform optimally for a given image. For example, the router network is fairly densely connected, and each window in the image is rotated using bilinear interpolation. Estimating and correcting for rotation is a large part of what makes the algorithm so demanding. If we know that faces are near-vertical, we can disable the rotation estimation. This will greatly reduce running time and reduce false positives.

Furthermore, most of the windows scanned are in the lower parts of the image pyramid. If we know the face is of a certain size, we can restrict the search to the top n levels of the pyramid, with only a fraction of the windows scanned. If we know the lighting conditions we can adjust the texture thresholding to discard as much as possible.

4 Face recognition

4.1 An overview of the algorithms

The first algorithms for face recognition, developed in the 60s were only semi-automatic. The coordinates of facial features were found manually, and the relative distances between these points were compared to stored templates. In 1988, Kirby and Sirovich applied principle component analysis to face recognition, and demonstrated that less than a hindered values were required to code a face [18], an approach since refined by Turk and Pentland [32]. Their approach is considered a milestone, as calculating the eigenvectors is a fairly computationally demanding. However, once the eigenvectors are found, projecting into face space is simply a matter of matrix multiplication. The primary advantage of the eigenface method is the system's speed and efficiency. The eigenface approach reduces the amount of data needed to identify an individual to 1/1000th of that in a full sized image.

Unlike object recognition in general, an individual face can change considerably over time or situations. One can for example grow a beard or simply change facial expression. All these challenges come in addition to the fact that the initial differences between faces can be relatively small. Compared to face detection, recognition is therefore much more difficult. All the same complicating factors are involved, but rather than discriminating between classes of objects, we are now comparing instances within the same class, and as a consequence we are much more sensitive to noise and distortions. But while recognition is more difficult than detection, the algorithms used are not vastly different, and the old distinction between feature based and image based approaches still holds.

4.1.1 Feature based methods

Feature based methods attempt to differentiate individuals by analysing their facial features. Faces vary in many respects, from the distance between the eyes, to the size of the nose. Furtermore, many of these do not vary, even over different expressions. If for the exact coordinates for a set of facial features can be found, then ther interrelation may specify the identity of the individual. As noted above, in the earliest attempts at semiautomatic reognition, the coordinates were extracted manually. Today, because these approaches require a very high level of presicion, high resolution images of 3D maps are often used. An elastic graph may then be fit to the 3D model, and compared with the stored template. Such a method is not

limited to frontal or vertical views.

4.1.2 Image based methods

As with detection, image based methods simply compare two surfaces in one or more mathematical, without explicitly extracting facial features. The predominant modern approaches are *Linear Discriminant analysis*, *principal component analysis* and *neural networks*. Linear discriminant analysis is a statistical approach aiming to minimize the within-class variance, and maximize the between class variance.

4.1.3 Recognition by eigenfaces

Turk and Pentlands eigenface approach treat individual faces as columns in a "face matrix". The principal components of this matrix can be calculated, and new faces can be projected along these bases to and the coordinates in "face space" determine the similarity to faces in the original matrix.

More formally, we start with a set of images Γ_1, Γ_1, ..., Γ_M. From these we calculate the average image, given simply by:

$$\Psi = \frac{1}{M}\Sigma_{n=1}^{M}\Gamma_n$$

This "mean face" is then subtracted from each of the original images. Each image is then transformed from N×N to N^2 vectors, and these are placed as columns in the face matrix. The covariance matrix is given by:

$$C = \frac{1}{M}\Sigma_{n=1}^{M}\Phi_n\Phi_n^T$$

Given that

$$A^T A v_i = \lambda_i v_i$$

Turk and Pentland multiply both sides with A, which gives us:

$$AA^T(Av)_i = \lambda_i(Av)_i$$

$$Cv = \lambda v$$

Reconstruction of the faces is done by projecting new vectors into "face space"

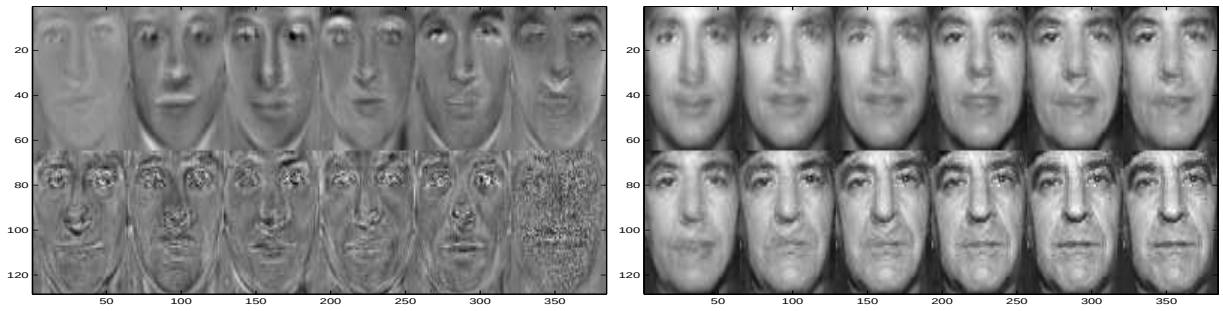

Figure 23: *Left:The 6 first and 6 last eigenfaces, Right:Reconstructing a face using an increasing number of eigenfaces (+2 per iteration)*

(they are transformed to their eigenface components).

$$\omega_k = u_k^T(\Gamma - \Psi)$$

Γ is the test image, Ψ is the training image, and u_k is the k'th eigenvector of the trainingimages. These make out the vector:

$$\Omega^T = [\omega_1, \omega 2 \ldots \omega_M]$$

For a given test image we can calculate the euclidean distance to the training images by:

$$\epsilon_k = \parallel \Omega - \Omega_k \parallel$$

$\min(\epsilon_k)$ is the closest match, and if it is less than a set threshold, the test image is classified as individual k.

Figure 23 demonstrates the reconstruction of a face given an increasing number of eigenfaces[23]. We can see that the first eigenfaces contribute considerably to the reconstruction, .

[23]the images are taken from the MATLAB prototype described in section 3.3

5 Implementation details

Implementing all the libraries from scratch has been very time consuming, and thoughts regarding why this was done and whether it was a good idea can be found in chapter 7. In this section I merely want to sketch how the code underlying the system is organized, and mention some of the more important design decisions. The final build contains approximately 17000 lines of C code, although some of this code is no longer in use. It is split into four main libraries. These are *neural networks*, *genetic algorithms*, *general image processing*, and *TIFF handling*. All available online at `heim.ifi.uio.no/nikolaic/hovedoppgave/`, both as raw code, and documented with doxygen [33]. The algorithm is very computationally demanding, and MATLAB was neither fast enough nor sufficiently flexible to be the main implementation platform. Therefore the system was implemented in C, although prototypes for various routines were first made in MATLAB, and Perl was used extensively for benchmarking and handling various text files, such as those specifying network topology.

For the linear algebra in the eigenface recognition, Gnu Scientific Library (GSL) was used. This is an extensive open source library providing a wide range of mathematical routines[24].

5.1 TIFF reading/writing library

The tiff library was made as we needed an efficient image structure that could be easily integrated with the neural network code. Most importantly, it had to be able to contain HSI encoded images, handling integer values, and not merely unsigned char. The current version is only able to read/write uncompressed images in colour or grayscale. The library also handles the construction of image pyramids, copying and initializing images, and a host of helper functions. The central structure around which this library is organized is the image:

```
typedef struct
{
    int type; /* 0==RGB,  1==HSV  2==Grayscale*/
    int x_dim, y_dim; /* Image dimensions. x=horisontal, y=vertical */

    /* ============= */
    /* For HSV images */
    int ** scan_lines_H;
    int ** scan_lines_S;
```

[24]GSL can be downloaded at `http://www.gnu.org/software/gsl/`.

```
    unsigned char ** scan_lines_V;

    /* ============= */
    /* For RGB images */
    unsigned char ** scan_lines_R;
    unsigned char ** scan_lines_G;
    unsigned char ** scan_lines_B;
} nik_image;
```

5.2 Image processing library

The image processing library takes care of the general image processing operations. These include colour space transform, histogram equalization, colour thresholding, calculating various image textures, filtering, image rotation and scaling. This library also contains all remnants of the model based localization approach, such as the highpass-filtering of interior skin-colour regions, various histogram functions, clustering algorithm, code for fitting triangle and elliptic models to skin colour areas, etc.

5.3 Neural network library

The neural network library allows us to easily create multilevel perceptrons, with full control over topology. By far most of the execution time of the algorithm is tied up in evaluating network response to the sliding window. It is bound up in a relatively tight amount of code, and it is therefore important that this is efficiently implemented. More time than I am willing to admit has been spent trying to optimize the neural network library.

We decided to implement our own NN code, after trying out a set of open source libraries and finding them either so simple that they did not provide the flexibility needed (ex in network topology), or so advanced, that it would be equally time consuming to get to know the code, as to build it from the ground up. Or so we thought.

Also, our library was developed to offer the required functionality, high flexibility in terms of topology and be fully integrated with the genetic algorithm routines.

Among the functions implemented in the NN library are routines to construct multilayer perceptron from files, train networks by backpropagation, scan images, export false hits into bootstrap vectors, construct neural networks from bit chromosomes etc.

The basic structures include the following:

```
struct neuron
{
  int index, level;
  struct neuron * next_neuron;
  struct weights * weights_to;
  struct weights * weights_from;
  double sum;                   /* Temp input variable */
  double activation;            /* Outgoing activation  */
  double delta;                 /* Target - output      */
  double delta_weight;
  double weight;
};

struct weight
{
    double strength;
    struct neuron * to;
    struct neuron * from;
    double strength_last; /* For momentum calculation.  */
};

struct weights
{
  struct weight * this_weight;
  struct weights * next;
};

struct network
{
  struct neuron *in, *hid,*out;
  int in_nr, hid_nr, out_nr;
  int weight_nr;
  unsigned short  train_vectors;
  unsigned char * in_vectors, * out_vectors;
  unsigned short  test_vectors;
  unsigned char  * in_test_vectors, *out_test_vectors;
  double error;
};
```

However, while networks based on these structures are flexible and intuitive, they are not efficient. Networks with many edges will take up a lot of memory, and cache size will quickly become a limiting factor. For example, in the early versions, using a router network increased the running time up to a factor of 10, although this network contains only about three times as many connections. Therefore, before traversing the image pyramid, the network is transformed to an array structure, with a compact linear memory layout. This greatly reduces the running time, and a single evaluation of the detector network can be carried out simply as:

```
/* ============  Read 20x20 window into input layer ============ */
for (i=0; i<d1in; i++)
```

```
  { dtmp=l_table[*((unsigned char*)(&eq_window[0][0]+i))];
    *(d1i_activation+i)= dtmp; }

 /* ============ Input to hidden layer ============ */
 for (wi=0,i=0; i<d1ih_weights; i++)  /*
   { **(d1ih_weight_to+i)+=(**(d1ih_weight_from+i))*(*(d1ih_weight+i)); }

 /* ============  Sigmoid output of hidden layer  ============*/
 for (i=0; i<d1hid; i++)
    { dtmp=*(d1h_activation+i);
    *(d1h_activation+i)= 1.0/(1.0+exp(-(dtmp))); }

 /* ============  Input to output layer  ============*/
 for (wi=0, i=0; i<d1ho_weights; i++)
    { **(d1ho_weight_to+i)+=(**(d1ho_weight_from+i))*(*(d1ho_weight+i)); }

 /* ============  Sigmoid of output layer  ============*/
  dtmp=*(d1o_activation);
   *(d1o_activation)= 1.0/(1.0+exp(-(dtmp)));
    response =  *(d1o_activation);
}
```

Another widely employed method of optimization is the use of precalculation. One example of the use of precalculated results are lookup-tables for input-layer response to pixel intensities. There are only 255 possible distinct values that the input layer can receive. These must first be shifted to the [-6.0,6.0]. The sigmoid function of the scaled pixel values can be looked up directly for a table rather than having to be calculated for every pixel. Other precalculations include, sin/cos values for window rotation, and $(X^T X)^{-1} X^T$ for the linear light model.

5.4 Genetic algorithm library

The basic structures in the genetic library are gene-pools, organisms, and bit-chromosomes. All the genetic operators introduced in section 2.4 can be found here.

6 Benchmarking

In this section we present results from benchmarking of the system. Although automatic recognition necessarily requires accurate localization, we have chosen to evaluate these two steps of the algorithm separately. This was done because a lot of effort was put into making localization robust even under difficult conditions such as sharp and directional lighting, occlusion, rotation and multiple faces at different scales in the same image. However, the widely used datasets for recognition almost exclusively contain single vertical frontal faces under ideal lighting and against a homogeneous background. To tax the localization sufficiently, we used a separate benchmarking set for localization.

6.1 Face localization

In benchmarking the face localization algorithm, we wished to shed light on the following issues:

i) How well is the algorithm able to localize faces?

ii) How does topology of the networks influence performance.?

iii) How large gains in speed can be achieved using colour/texture thresholding, and how does thresholding impact the number of correctly localized faces?

iv) How demanding is the estimation of rotation, and how much does it degrade performance in localizing vertical faces?

When benchmarking, we wish to evaluate how well an algorithm/system performs on a representative problem set. Before running the benchmarking trials, two issues must be clarified. *How is the benchmark set selected*, and *how is performance judged*. While necessary to compare different algorithms, these issues are rarely discussed in face localization literature.

6.1.1 Benchmarking set

Unlike recognition, there are fewer established benchmarking sets for face localization, and often authors simply benchmark a set of images they themselves have collected. One of the few sets available is that of Rowley et.al., and consists of 130 grayscale images, containing in total 507 faces (the bulk of which are present in group photos). The set contains images of frontal faces rotated in the plane, and where

Figure 24: *Output from face localization trials.*

most are set against a cluttered background. The database can be downloaded from `http://www.ri.cmu.edu/projects/project_419.html`.

The Rowley face set is challenging, but contains only grayscale images. To benchmark our implementation, we therefore selected a subset of the Rowley database, and added colour images found on the Internet. The images were selected to be challenging and representative of natural variation in face appearance. Faces were mostly set against a cluttered background, some had difficult lighting, rotation in the image plane, occlusion, and variations in size and expression. The final set consisted of 106 colour images containing in total 269 faces, and 75 grayscale images with a total of 186 faces. Image dimensions ranged from 150×200 up to 900×600, with the majority being around 400×500. None of the images used during training of the networks were included in the benchmarking set. The final set and the result of localization benchmarking can be viewed at `heim.ifi.uio.no/nikolaic/hovedoppgave/`. The output images from all benchmarking trials are available upon request.

6.1.2 Scoring hits and misses

There are a multitude of published algorithms for face localization, most claim excellent performance and present convincing statistics. Authors usually present examples of successful localizations by having the system draw a bounding box around an image region. However, a general problem in evaluating competing approaches is the lack of a standardized set of criteria for how accurately a facial region must be circumscribed in order to be declared a successful hit [21]. Kriegman suggests the evaluation of a localization algorithm should depend on its purpose. [19]. If a system

is to count the number of individuals present or track faces, it may not need exact coordinates of the facial region. However, as illustrated with our initial MATLAB prototype, the eigenface approach is very sensitive to poor segmentation, and many "successful hits" may need substantial pre-processing before they can be presented to the classifier. In our benchmarking, we therefore adopted fairly strict criteria for successful localization. The face region should be circumscribed sufficiently well for recognition to be carried out successfully. The bounding box should therefore just cover the inner facial region, not extending higher than just above the eyebrows, and not lower than the chin. Furthermore, error in rotation estimation must not be greater than 10 degrees, or the response will be declared a miss. Hits and misses were scored manually.

Numerous parameters can be changed to drastically reduce the number of false positives at some expense of true positives. However, as the ultimate purpose of the system was recognition, false negatives are more damaging than false positives, and a fair amount of false positives was accepted. Furthermore, while several ways of selecting among overlapping face hits have been implemented, no such selection was performed during the final benchmarking. A single face will therefore usually give rise to several adjacent hits, and the criteria for success is that at least one of them is bounded as described above. In the same way as multiple hits on the same face are counted as one true positive, overlapping false positives (similar size/rotation) are counted as a single false positive.

6.1.3 Training data, algorithm parameters and test system

Training of the neural networks has been described more extensively in section 3.5.3, but it may be helpful with a briefly review before proceeding. To train the detector network 192 faces were manually segmented, mirrored, translated and finally down-scaled to 20×20 to form a vector of 1214 faces. 11808 non-faces were randomly extracted from image pyramids of satellite photos. After the networks were trained with these vectors, 1500 false positives from trial runs on satellite photos were added as non-faces to the training set before the networks were re-trained. All detector networks were trained on the same input.

The router networks were trained with 334 faces rotated n$\times$10 degrees in MATLAB, giving a total rotation training vector of 12024. Rowley et. al. used a fully connected router network, and a detector networks whose topology is described in section 3.4. We compared this setup with combinations of either a router network

Table 1: Benchmarking of face localization

Type	Hits	Misses	False positives	Time used (s/im)
(1) Evolved router and evolved detector				
Colour	209 (77.7%)	60 (23.3%)	212	1509 (14.2)
Grayscale	162 (87.1%)	24 (12.9%)	189	1205 (16.0)
(2) Fully connected router and evolved detector				
Colour	208 (77.3%)	61 (22.6%)	206	1720 (16.2)
Grayscale	154 (82.8%)	32 (17.2%)	195	1394 (18.6)
(3) Evolved router and Rowley detector				
Colour	218 (81.0%)	51 (18.9%)	256	1487 (14.0)
Grayscale	161 (86.6%)	25 (13.4%)	207	1181 (15.7)
(4) Fully connected router and Rowley detector				
Colour	205 (76.2%)	64 (23.8%)	255	1713 (16.3)
Grayscale	156 (83.9%)	30 (16.3%)	210	1358 (18.1)

with evolved topology, a detector network with evolved topology, or both. During benchmarking, all 181 images were scanned down to the second layer of the pyramid, meaning that the smallest detected face would be of size 28×28 pixels. The benchmarking was performed on a 64bit AMD Athlon 3000 running Gentoo linux 2.6.11-r1.

6.1.4 Benchmarking results

Localization rates across different network topologies are presented in table 1. Configuration 4 uses the same topology as Rowley et.al. on both the router and detector networks. While all combinations of evolved and standard networks learned the task well, and differences in success rates were small, configuration 4 was the one with overall poorest performance. The best performance was achieved with configuration 3, an evolved router network and three standard detector networks. This configuration was able to correctly localize 81% of faces in the colour images and 86.6% in grayscale, while having the shortest running time. The evolved router network is more sparsely connected than the fully connected, and this resulted in a speed increase of 13% compared to configuration 4.

The difference in the number of false positives was not affected considerably

Table 2: Effects of colour and texture thresholding on face localization

Type	Hits	Misses	Time used (s)	Win scanned	Win discarded
Colour and texture threshoding enabled					
Colour	209 (77.7%)	60 (23.3%)	1509 (14.2)	11.71E6	31.06E6
Grayscale	162 (87.1%)	24 (12.9%)	1205 (16.0)	9.52E6	12.99E6
Texture threshoding disabled (colour enabled)					
Colour	212 (78.8%)	57 (21.2%)	3079 (29.0)	23.35E6	19.41E6
Grayscale	162 (87.1%)	24 (12.9%)	2933 (39.1)	22.51E6	0
Colour threshoding disabled (texture enabled)					
Colour	209 (77.7%)	60 (23.3%)	1998 (18.8)	15.63E6	27.15E6
Texture and colour thresholding disabled					
Colour	212 (78.8%)	57 (21.2%)	5885 (55.5)	42.77E6	0

[a]All benchmarks performed with evolved network topologies

by network topology, and remained stable at around 2.5 per image. The rate of false positives can be drastically reduced by requiring a higher response in the cells surrounding a potential hit. Increasing the necessary local response from 13.9 to 17 resulted in 8 additional missed faces, while the number of false positives dropped from 189 to 85 on the graylevel set.

Table 2 shows the effect of colour and texture thresholding on localization rates and processing time. The first question of interest is how many faces are lost when the images are thresholded. Column 2 indicates that no faces in the grayscale set, and only three faces in the colour set were missed due to texture thresholding. Furthermore no faces were lost because of colour thresholding.

While thresholding had little effect on detection rates, the impact on running time of the algorithm was considerable. Without colour and texture thresholding, the processing of the colour images takes 5885 seconds. If only colour thresholding is enabled, processing time drops to 3079 s.,nearly half (52.3%). On the other hand, if only texture thresholding is enabled, processing of the colour images takes 1998 s., (33.8%). Rather than look at execution time, we can observe how many 20×20 pixel windows in the pyramids we can ignore. In the colour set, 19.4E6 windows (45.4%) can be discarded solely on the basis of colour, 27.2E6 (63.3%) solely on texture, and 31E6 (72.6%) if both colour and texture are used. As processing time

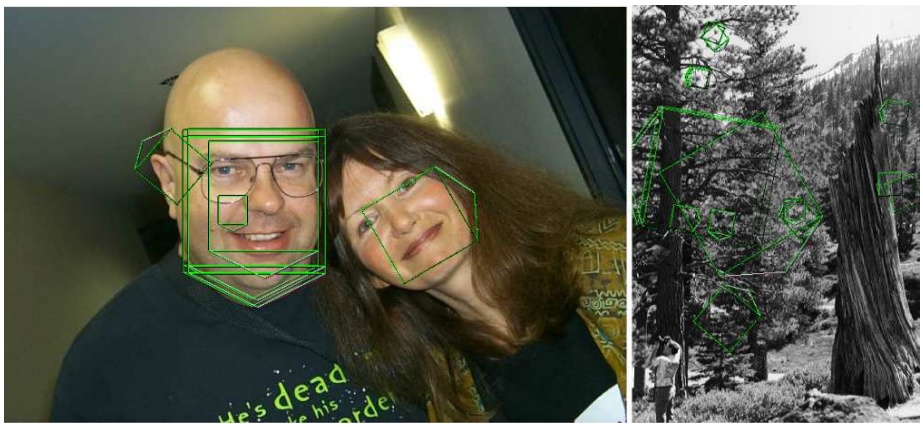

Figure 25: *False negative (left), False positives (right)*

is almost linearly related to the number of windows scanned, thresholding on both colour and texture reduce the time it takes to scan the 106 colour images by 74.4%. It is interesting to note that with the skin-colour model we used, the largest gains in speed were achieved through thresholding on texture, not colour. We therefore attempted to be more restrictive in classifying skin-colour, but this resulted in an unacceptable level of false negatives. In the grayscale set, texture thresholding resulted in a 57.7% reduction in running time.

The slight difference in the number of windows discarded due to texture in colour and grayscale images is likely an artifact of the images across the sets. Some of the grayscale images have especially cluttered backgrounds, and texture thresholding has less effect. From table 1 it appears that more faces are localized in the grayscale images. However, it seems that also this difference is an artifact of the images in these sets. The detection networks are colour blind, and disabling skin colour thresholding did not increase the number of detected faces. We can only conclude that the small difference in performance between these two test sets is due to the colour images somehow being more difficult.

Besides reducing processing time, texture thresholding had an enormous effect on the number of false positives, and when thresholding was disabled, there were so many false positives that they could not be counted manually. While more than two false positives per image may seem a lot, consider the number of windows actually scanned. If 212 false positives result from scanning the 11.7E6 windows in the colour set, this amounts to an error rate of 0.0018%.

Estimating rotation may lead to deterioration in detection of vertical faces. We also knew from the testing we had done during development that rotation estimation is very costly, and requires a lot of processing. To explore these two issues, we ran the

Table 3: Examining the effect of rotation estimation on vertical faces

Type	Hits	Misses	Time used (s/im)
Rotation estimation enabled			
Colour	180 (81.82%)	40 (18.18%)	1509 (14.2)
Grayscale	115 (94.2%)	7 (5.7%)	1205 (16.0)
Rotation estimation disabled			
Colour	195 (88.6%)	25 (11.4%)	282 (1.8)
Grayscale	116 (95.0%)	6 (4.9%)	215 (2.9)

algorithm on the same test set as earlier, but with rotation estimation disabled. We then counted how many of the vertical faces[25] were localized correctly. The results are presented in table 3. The most striking result is how much the running time is reduced when rotation estimation is disabled, as it drops to less than a fifth. How can we explain such a vast difference in performance? The first factor accounting for this is the router network. This densely connected network is run on every non-masked window in the pyramid. After a window is rotated back the estimated number of degrees, it is compared to the mean training face. If the distance is too large the algorithm branches without running any of the detector networks. Furthermore, the detectors contain about half as many connections as the evolved router network, and very rarely will all three networks be run. This is because the algorithm branches whenever the threshold for detection can no longer be achieved. In short, the densely connected router is run at every non-masked location, where as the detectors are more sparsely connected and rarely are all of them run. The second factor contributing to the cost of estimating rotation is the process of actually rotating back every window before feeding it to the detectors. Each window is rotated using bilinear interpolation. In the average image, 10E5 windows must be rotated. To investigate the cost of using bilinear interpolation during rotation, we ran the benchmarks again, but with a nearest neighbour rotation. This resulted in a reduction in processing time of 16%.

[25] Vertical faces had a tilt of less than 10 degree.

6.1.5 Discussion and comparison to other systems

As already mentioned, reducing false positives was not a priority, and the results indicate that every image has on average 2.5 false hits. The number of such false positives can be substantially reduced by with only a small deterioration of true positives, by setting the required response rate for surrounding locations higher. On the other hand, if texture thresholding was disabled, an unacceptable large number of false positives was observed.

Given that no faces were lost due to colour thresholding, it is worth checking whether the allowed colour boundaries could be made more restrictive, and optimization gain greater. The skin colour threshold were on our implementation fairly liberal, because the benchmarking set covered a wide range of lighting condition. If we were to implement this again, we would let the algorithm also learn the best Hue/Saturation values from a similarly varied training set.

As for texture, although variance and entropy reduced the number of windows need to scan by as much as 2/3, it is worth conducting a more rigorous study of which other textures coluld be used. Both in order to reduce execution time, but perhaps even more importantly, to reduce false positives [26].

As explained above, comparing these results to other systems is difficult, as neither datasets or criteria for scoring are standardized. In a rotated face set, Rowley's final algorithm correctly localized 85.7% of the faces, with only 15 false positives [23]. The recognition rates are similar to those we have achieved, although we experienced higher levels of false positives. The reduction in processing time is hard to compare to Rowleys setup, as they do not give the running times in their article. However, in an earlier paper, vertical face detection carried out on an 200MHz R4400 SGI Indigo 2 is said to use 383 seconds to scan a 320×240 image. Mean running times from our vertical detection benchmarking are at 1.8 seconds for colour images and 2.9 for grayscale. Although the colour images in our set were processed more than two hundred times faster, it is impossible to parse out the effects of the network code, colour/texture thresholding and machine architecture. Suffice to say, real-time vertical neural network based face localization may soon be possible, even with a mainstream general purpose processor.

[26] We also briefly experimented with some textures intended only to reduce false positives, such as different types of symmetry

6.2 Face recognition

In benchmarking the face recognition algorithm, we were primarily interested in:

i) Whether the faces in the recognition set were detected, and how the accuracy of localization impacted recognition rates.

ii) Whether manual or automatic extraction of the face in the training image resulted in different recognition rates across the test images.

iii) How recognition deteriorates as difference from the training image increases.

6.2.1 Benchmarking set

Numerous test sets are available for benchmarking face recognition algorithms. However, not all are easy to get hold of, and most of them have certain limitations. These include among other things few images per individual, or images that are poorly standardized[27]. Because few recognition algorithms rely on colour, many of the sets are grayscale. As they are used for testing recognition, localization of faces in these images is usually very easy. The background is homogeneous, lighting is good, and the faces are usually roughly the same scale and orientation.

After evaluating several datasets (among them FERET [20], Yale set [38] , BioID [3], Stirling [29] and others), we eventually settled on the *Georgia tech face database* [30]. This image set it is freely downloadable, consists of colour images with at least marginally cluttered and partially skin-coloured backgrounds. Faces are of good resolution, but do not fill the entire image surface. This way the images present some challenge to localization. The dataset is also sufficiently large, consisting of 50 individuals, and contains 10 images per person. While the set has all these positive properties, it is not as well standardized as some of the more established sets, and different individuals have different expressions and minor variations in facial orientations. Our final sample consists of 4 images of 40 different individuals. Image 1 will be used for training. Image 2 was selected to be as similar as possible to the first (although none of them were identical). In image 3 and 4, the face becomes progressively different from the training image. They are different in facial expression, scale or occluding objects such as glasses. All images are of dimension 640×480.

[27] varying in size, posture, tilt or expression across individuals

6.2.2 System setup

Based on our initial MATLAB prototype, we decided to use a slightly higher resolution on the eigenfaces, increasing the dimensions from 64×64 to 88×88. It is well established that low resolution of the face images significantly limits the performance of eigenface based recognition systems [10]. As all face regions will be downscaled to these dimensions prior to classification, there is no need to traverse the lower levels of the image pyramid (below layer 7). Furthermore, as all the faces in the test set are vertical, and rotation estimation is so demanding, it was disabled during benchmarking.

Given that the eigenface approach is sensitive to changes in translation, scale and orientation, we hypothesized that we may increase recognition rates by storing several templates of each individual in the database. We therefore made two test sets. In the first, one optimal facial region was manually extracted for all the images. In the second set, the localization algorithm extracted the faces automatically. As neighbouring locations usually all cross the detection threshold, the second database contained several entries for each individual, 252 in all.

Because a single face will produce several hits, each of these was classified in the hope that at least one would be sufficiently well segmented to be correctly recognized. Occasionally, different boundings of the same face will be classified as different individuals, and we must somehow select one. In our trials, an individual was recognized if the norm between the coordinate vector in eigen-space of the stored template and the new vector was less than 2000. If more than one individual was recognized in the same area, the one with the lowest distance to the stored template was selected. In these trials, the 150 eigenvectors with the largest eigenvalues were used.

6.2.3 Benchmarking results

Table 4 gives the results from the recognition benchmarking. All faces in the 160 images were localized correctly, and the mean time used for all images was 289.6 seconds (1.8 s/im). The perfect localization rate is not surprising given the performance on the more challenging set in section 6.1.

When the templates for the face database were automatically extracted from the training image (1), all individuals were subsequently recognized from the same image. However, already in the first test image recognition rates drops to 80%. As the difference between the faces increases from training to test, recognition drops

Figure 26: *Output from face recognition. Green squares surround localized faces and red squares indicate a recognized individual*

Table 4: results from face recognition

Image	Localization failures	Correctly classified	Incorrectly classified
(1) 150 eigenvectors of the 252 automatically extracted			
Training image	0	40 (100%)	0 (0%)
Test image 1	0	32 (80%)	8 (20%)
Test image 2	0	25 (62.5%)	15 (47.5%)
Test image 3	0	20 (50%)	20 (50%)
(2) Manually determined face boundaries			
Training image 1	0	35 (85%)	5 (12%)
Test image 1 2	0	26 (65%)	14 (35%)
Test image 2	0	15 (37.5%)	25 (62.5%)
Test image 3	0	15 (37.5%)	25 (62.5%)

further to 47.5% and 50% in test images 2 and 3. While these results show that a pure eigenface approach is sensitive to changes in face appearance, the recognition rate achieved using several automatically segmented templates per individual was superior to that of manually segmenting a single example from the training set. One problem with using a sliding-window of fixed size is that if the training face is manually segmented in such a way that the dimensions fall halfway between two levels in the pyramid, then the later automatic segmentation may look quite different to the stored template.

6.3 Comparisons to other systems

State of the art face recognition systems achieve a recognition rate of 98% on very similar images. However, deterioration in recognition rate is high when the images are taken over time and under different lighting conditions [41].

7 Conclusions and personal comments

The goal of this project was to *implement* and *evaluate* an automatic face recognition system. After a model-based face localization algorithm was abandoned, we switched to a neural network based approach that has previously demonstrated robust performance. This approach is very computationally intensive, and estimating rotation is the most demanding aspect. Our contribution to the localization algorithm is primarily to reduce the processing time by:
i) using a genetic algorithm to evolve a more sparsely connected router network.
ii) limit the search space by thresholding the input image on skin colour and texture, also reducing false positives

An eigenface based algorithm was used for face recognition. Our contribution was to have the system automatically extract multiple and differently segmented faces from the training image, rather than manually segmenting a single face.

Evaluation of the localization algorithm demonstrated that it is generally robust, able to correctly and accurately segment faces despite variations in lighting, rotation, scale, facial expression and occlusion. Furthermore, although the difference between network topologies was small, the evolved router topology improved speed by roughly 13%, while improving the localization rate compared to Rowleys fully connected topology by 3%. Texture thresholding reduced the running time of the algorithm by almost two thirds in both colour and grayscale images, while skin colour thresholding independently reduced the running time by 45%. In colour images the combined effect was a reduction in processing time of 75%.

Benchmarking of the recognition algorithm demonstrated that eigenface-based face recognition is very sensitive to variations in segmentation relative to the stored template. However, because automatic segmentation is likely to vary somewhat compared to a manual segmentation of the same face, performance of the eigenface based approach can be improved by storing several slightly differently segmented templates per person.

The program was implemented in C, and new libraries for neural networks, image processing, TIFF reading/writing and genetic algorithms were written to make the system fast and flexible .

7.1 On a more personal note

Before concluding this work, I would like to reflect on a more personal note on different aspects of the project, and on different levels of abstraction, the *code* written, the *algorithm* implemented and the general pursuit of *face localization and recognition*.

Writing the code;. All of the code, except for standard C and linear algebra libraries was written from scratch. This was largely motivated by a desire to have complete control in making the system as efficient as possible. Consequently, a lot of time was spent designing, implementing and debugging the various components. Once a working system had been implemented, further time was spent optimizing the code. As an example of this, consider that preliminary results from the localization algorithm were presented at the NOBIM 2004 conference [6]. The 2004 version of the code could spend up to 5 minutes analyzing a single large image. When the fastest implementation dating from that time was re-run on the final grayscale image set, it used 3981 seconds, 337% more than the final build. This difference is solely due to optimization of the source code. However, while the time spent implementing and optimizing the code has constituted a large part of my work on this thesis, few of the design decisions and optimizations performed are individually important enough to be presented in the written document.

Another often problematic consequence of not using existing source code was trying to understand what was wrong the many times when things didn't work as we had anticipated. Was it because the many parameters were set incorrectly? Was the training data poor? Or is there simply some obscure pointer bug deep in the network handling code. The upshot of writing all the NN, image processing and GA software was of course that it has been immensely educational. Still, if I were to start over today, I would probably have done things a little differently. To begin with, I would have chosen an object oriented platform (C++), as many elements in the algorithm from the individual neurons, networks to images, are best understood as objects with associated operators, rather than solely as passive memory structures. But admittedly, in addition to wanting to reinvent the wheel at every point, part of the reason for programming everything myself was not being used to and trained in looking for existing open source code.

The algorithm; Before starting work on this project I knew that automatic

recognition is far more difficult than people without any experience in machine vision and image processing often think. This did not prevent me from being surprised at how little deviation from ideal conditions was enough for our initial model based localization approach to fail. Later I was surprised again, this time positively, at how robust the neural network based approach was. And although neural networks have been mentioned in passing at various points through my education, it has often been with some reservation. So, although I recognize that caution has to be exercised not to resort to NNs as a least effort black-box approach when better analytic tools are available, I have come to believe that at least some of what I always felt was a slightly tarnished reputation in academic circles is unjustified.

While the algorithm presented in this thesis currently does not work at real time, another benefit of the neural network based approach is the almost limitless paralelizability possible. In genuine neural network, information flows along all the connections parallel and in continuous time. It is almost absurd that what in reality is such a vastly parallel system can be meaningfully simulated in a single digital processor. Even disregarding the commercial availability of dedicated NN chips that would harness some of the genuine parallel processing [31], the algorithm would benefit from as many more conventional processors you would be willing to throw at it. Each of these could independently traverse and evaluate windows at different locations in the pyramid.

A different interesting quality of the approach is how results can often be improved by combining the output of several networks trained with different initial weights or different topologies. In out implementation we used 3 detectors where previous authors used two.

The use of genetic algorithm may seem somewhat misplaced. After all, the gain was fairly modest compared to those achieved simply through thresholding. Our motivation for spending so much time and effort in the router was because in an earlier version of the neural network code less optimized on memory, the overwhelming bottle neck in terms of speed was the size of the router network. Somewhere between the size of the detector network and the router network, cache limits were breached. Consequently, a router network could be up to two orders of magnitude slower than a detector network, even though it is less than four times the size. The overwhelming priority was therefore at the time to reduce the size of the router network in a well reasoned way. However, in a later phase of development, a memory efficient version of the neural network code resulted in the time use of the router network being only a linear increase, as can be seen from the final benchmarking results.

On face localization and recognition; Face detection and recognition are for reasons outlined earlier, very difficult tasks. It is also an area with considerable research activity, and where the rewards, both financial and academic to those coming up with a fast and robust algorithm are potentially very large. The bottom line is that a lot of highly skilled people around the world are working on the task, and most of them with better financial backing than your average student. And despite this, robust recognition and perfect localization is still a challenge. So while we almost by definition never can actually reach the goal we have set, it is always possible to do just a little better on the benchmarking set. Just one more recognized individual or one less false positive. As the source code will reveal, there are many parameters that can be tweaked in order to improve performance, and to make a long story short, an unhealthy amount of tweaking has taken place.

7.2 Conclusion

The work on this project has been challenging, but ultimately extremely educational and rewarding. I feel I have had many programming challenges, but in addition to the purely technical, topics such as biometrics, neural networks and genetic algorithms have also been philosophically satisfying. Face recognition has been a subject that not only I have found fascinating, but that has engaged others, both within and outside of the field of computer science.

8 Appendix: HS clustering algorithm

Our first attempt at localizing faces in images made use of a face model, according to which faces could be described as triangles. Essentially, the system tried to fit isosceles[28] to the center of mass of the most prominent highpass filtered objects of the interior[29] of skin-coloured areas. These objects represent the usual rapid intensity transitions associated with the eyes and mouth. Consequently, at most *one* face would be localized within each skin colour region. In testing the system it became apparent that often a single skin coloured region merged two or more faces, so the algorithm would fail, or at best only localize a single face. We therefore developed a crude clustering algorithm to split merged skin-colour regions. While

[28] triangles with two sides of equal length

[29] we used only the interior, so as not to be confused by transitions to the background, hairline etc.

Figure 27: *From the left: Input image, skin colour thresholded, result after histogram clustering and cleaning. While they are merged in one skin colour region after thresholding, all faces separated after clustering.*

the model-based approach was abandoned in favour of a more robust, albeit slower neural network based approach, the clustering algorithm worked sufficiently well to deserve some mention.

Algorithm 3 2D hue/saturation Clustering algorithm

```
img = load input image()
img_HSI = transform_RGB_to_HSI(img)
2d_hist = calculate_2d hue/saturation histogram (img_HSI)
2x2 mean filter (2d_hist)
threshold 2dhist on 0.9 stdev
merge regions separated by 1 pixel
Assign each location in 2dhist to the closest peak based on euclidian distance
```

Algorithm 3 describes the clustering algorithm. Essentially, the image is transformed to HSI, and the two-dimensional Hue/Saturation histogram is computed. The histogram is thresholded so that cells more than 0.9 standard deviations above the mean level are set to 1 and the rest are set to 0. After merging close objects, each index in the 2d histogram is assigned to the peak that is closest by euclidian distance. Figure 27 gives an illustration of the effect of the clustering algorithm. Notice that while the original skin colour thresholding merged all three faces, these are separated in individual skin colour regions after clustering.

9 Appendix: Sample images from model-based localization

Figure 28: *Failure of a model-based approach*

Figure 28 illustrates failure in the model-based approach (top left) and output on the same image with the neural network based based approach (top-right). As can be seen from the images underneath, the clustering algorithm manages to split the skin coloured area in a way that separates all faces. However, the highpass filtering stage extracts objects other than the eyes and mouth, so the triangle fitting result in false positives. The neural network based localization is accurate, and segments the faces without too much background. The face on the left is missed as it is too much in profile.

10 Appendix: Sample images from face localization benchmarking

Figure 29: *Example images from face localization in the colour set*

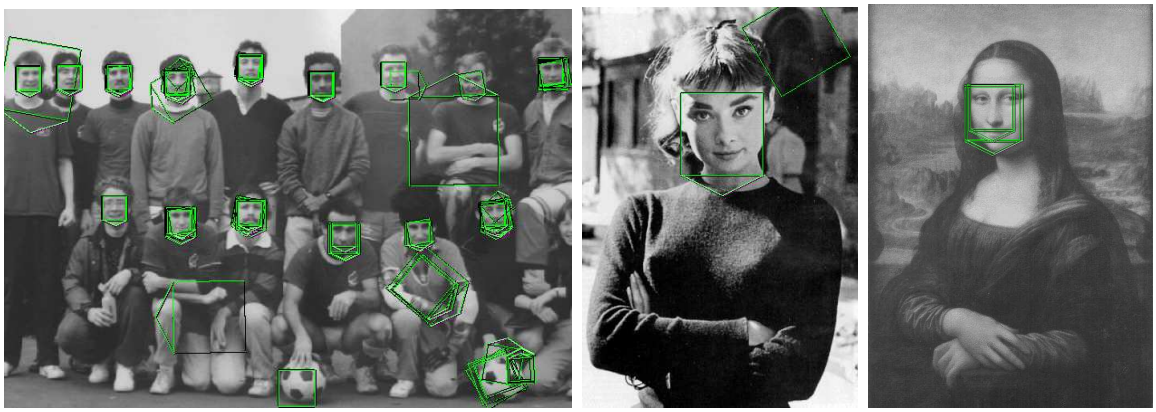

Figure 30: *Example images from face localization in the grayscale set*

References

[1] *TIFF 6.0 Spesification*, pages 1–121. Aldus Corporation, sixth edition, 1992.

[2] J. Barton, M. V. Cherkasova, D. Z. Press, J. M. Intriligator, and M. O'Connor. Developmental prosopagnosia: A study of three patients. *Brain And Cognition*, 51:12–30, 2002.

[3] BioID. Bioid face database. http://www.humanscan.de/support/downloads/facedb.php. Visited 30.07.2006.

[4] J. Branke. Evolutionary algorithms for neural network design and training. In *the first Nordic workshop on genetic algorithms and their aplications*, pages 146–163. University of Wasa, Findland, 1995.

[5] R. Carlson. *Physiology of Behavior*. Allyn Bacon, 9 edition, 2006.

[6] N. Czajkowski and A. Solberg. Optimized neural network-based face localization. . *In: Norwegian Conference on Image Processing and Pattern Recognition. Proceedings. NOBIM, Stavanger*, 2004.

[7] M. N. Dailey and G. Cottrell. Organization of face and object recognition in modular neural network models. *Neural Networks*, 12:1053–1073, 1999.

[8] K. Delac and M. Girgic. A survey of biometric recognition methods. *46th International symposium Electronics in Marine, ELMAR 2004*, pages 184–193, June 2004.

[9] I. R. Fasel and J. R. Movellan. A comparison of face detection algorithms. 2415:1325–1330,, 2002.

[10] K. Gunturk, Y. Altunbasak, M. Hayes, and R. Merseau. Eigenface-domain super-resolution for face recognition. *IEEE transactions on image processing*, 12(5):597–606, May 2003.

[11] E. Hjelmås and B. Low. Face detection: a survey. *Computer Vision and Image Understanding*, 83:234–274, 2001.

[12] I. Hsieh, K. Fan, and C. Lin. A statistic approach to the detection of human faces in color nature scene. *Pattern recognition*, pages 1583–1596, 2002.

[13] G. W. Humphreys and M. J. Riddoch. *To See But Not To See: A Case Study Of Visual Agnosia*. Psychology Press, 1987.

[14] A. Jain, J. Hong, and S. Pankanti. Communications of the acm. *Communications of the ACM*, 43(2):91–98, Feb 2000.

[15] A. Jain and M. J. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, March 1996.

[16] A. Jain and A. Ross. Multibiometric systems. *Communications of the ACM*, 47(1):34–40, Jan 2004.

[17] C. Lin and K.-C. Fan. A color-triangle based approach to the detection of human face. *Pattern recognition*, 34(6):1271–1284, 2000.

[18] K. M and S. L. Application of the karhunen-loeve procedure for thecharacterization of human faces. *Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990.

[19] D. K. Ming-Hsuan Yang and N. Ahuja. Detecting faces in images:a surveyu. *IEEE transactions on pattern analysis and machine intelligence*, 24(1):34–58, 2002.

[20] P. Phillips, H. Moon, S. Rizvi, and P. Rauss. The feret evaluation methodology for face-recognition algorithms. *IEEE transactions on pattern analysis and machine intelligence*, 22(10):1090–1104, 2000.

[21] V. Popovici, J.-P. Thiran, Y. Rodriguez, and S. Marcel. On performance evaluation of face detection and localization algorithms. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 1:313–317, 2004.

[22] H. Rowley, S. Baluja, and K. Takeo. Neural network based face. *Pattern Analysis and Machine Intelligence*, 20:23–38, January 1998.

[23] H. Rowley, S. Baluja, and K. Takeo. Rotation invariant neural network-based face detection. *Computer Vision and Pattern Recognition*, pages 963–963, June 1998.

[24] D. E. Rumelhart, G. E. Hintont, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[25] R. S., T. P., S. B., and B. A. Computationally efficient face detection. `http://research.microsoft.com/vision /cambridge/papers/romdhani_iccv01.pdf`. Visited 14.02.2004.

[26] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. *Computer Vision and Pattern Recognition*, 2:994–1000, 2005.

[27] F. Simion, V. Cassia, T. Curati, and E. Valenza. The origins of face perception: specific versus non-specific mechanisms. *Infant and Child Development*, 18(1):59–65, 2001.

[28] S. K. Singh1, D. S. Chauhan, M. Vatsa, and R. Singh. A robust skin color based face detection algorithm. *Tamkang Journal of Science and Engineering*, 6(4):227–234, 2003.

[29] Stirling. Stirling psychological face database. Available at: `http://pics.psych.stir.ac.uk/cgi-bin/PICS/New/pics.cgi`. Visited 10.06.2006.

[30] G. Tech. Georgia tech face database. http://www.anefian.com/face$_r eco.htm. Visited$30.07.2006.

[31] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin, and W. Wolf. Embedded hardware face detection. In *17th International Conference on VLSI Design*, 2004.

[32] M. Turk and P. AP. Face recognition using eigenfaces. *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference*, pages 586–591, June 1991.

[33] D. van Heesch. Doxygen; a documentation system for c++, c, java, objective-c, python, idl (corba and microsoft flavors) and to some extent php, c, and d. Available at: http://www.stack.nl/ dimitri/doxygen/.

[34] Y. Wang and B. Yuan. A novel approach for human face detection from color images under complex background. *Pattern recognition*, 34:1983–1992, 2001.

[35] S. Wiegand, C. Igel, and U. Handmann. Evolutionary optimization of neural networks for face detection. In *12th European Symposium on artificial neural networks (ESANN 2004)*. ESANN, 2004.

[36] Wikipedia. Biometrics, 2006. [Online; accessed 19-May-2006].

[37] Wikipedia. Neuron, 2006. [Online; accessed 31-July-2006].

[38] Yale. Yale face database. Available at: http://cvc.yale.edu/projects/yalefaces/yalefaces.html.

[39] M. Yang, D. Kriegman, and N. Ahuja. Detecting faces in images: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:34–58, 2002.

[40] X. Yao. Evolving artificial neural networks. In *Proceedings of the IEEE*, volume 87, 1999.

[41] W. Zhao, R. Chellapa, P. Phillips, and A. Rosenfeld. Face recognition: Aliterary survey. *ACM computing surveys*, 35(4):399–458, December 2003.

[42] Z. Zhongfei, R. Srihari, and A. Rao. Face detection and its applications in intelligent and focused image retrieval. *Proceedings. 11th IEEE International Conference on*, pages 121–128, November 1998.