Háskólinn
á Akureyri

Háskólinn á Akureyri

Viðskipta - og raunvísindadeild

Final Year Project Dissertation

2008

Nicholas Cudjoe

ABSTRACT

This document presents how a face recognition system can be designed with artificial neural network. Face recognition involves mapping of the images of a person's face to the identity of that person.

I have implemented a neural network that uses a back-propagation algorithm to recognize and classify four aspects of image of a person's face. These include the identity of the person, the expression of the person in the image, the orientation of the face in the image and whether the person sports sunglasses.

The final results was a MatLab built software application with an images provided by Dr. Tom Mitchell and utilizes the features of the images of the people taken with varying pose (straight, left, right, up) expressions (neutral, happy, sad, angry) eyes, i.e. wearing sunglasses or not and resolution.

The structure of the final software application is illustrated. Furthermore, the results of its performance are illustrated by a detailed example.

Keywords: Face recognition, Artificial Neural Network, MatLab

ACKNOWLEDGEMENT

I would like to thank my supervisor Dr. Tony Y. T .Chan, for his guidance and support and also Dr. Tangming Yuan for his recommendations in development of this project.

I wish to express my profound gratitude to my family for their continuous support and great love.

TABLE OF CONTENTS

LIST OF FIGURES

TABLE OF TABLES

INTRODUCTION

Human often uses faces to recognize individuals. Advancement in computing capabilities over the years now enables similar recognition automatically. Face recognition is a pattern recognition task usually performed on human faces. Humans are very good at recognizing faces and complex patterns. Even with passage of time, it does not affect these capabilities and therefore, it would help if computers become as robust as human in face recognition.

Face recognition system can help in many ways; face recognition can be used for both verification and identification. Today, recognition technology is applied to a wide variety of problems like passport fraud, human computer interaction and support for law enforcement. This has motivated researchers to develop computational models to identify faces, which are relatively simple and easy to implement.

PROJECT DESCRIPTION

The learning task here involves classifying camera images of faces of various people in various pose. Images of 20 different people were collected, including approximately 32 images per person, varying the person's expression (happy, sad, angry, neutral), the direction in which they were looking (left, right, straight, up), and whether or not they are wearing sunglasses. There is also a variation in the background behind the person, the clothing worn by the person, and the position of the person's face within the image. In total 640 grayscale images were collected, the scale of the image and this has 3 values: (1, 2, and 4) 1 indicates a full resolution image of (128 columns by 120 rows); with each image pixel described by a grayscale intensity value between 0 (black) and 255(white), 2 indicate a half resolution image (64 by 60); 4 indicates a quarter resolution image (32 by 30). The image data set was provided by Tom Mitchell and obtained from http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-8/faceimages/faces.

PROJECT OBJECTIVES

The objective of this project is to design a neural network that uses a backpropagation algorithm to recognize and classify three aspects of image of a person's face. These are: the identity of the image, the orientation of the face in the image and whether the person sports sunglasses or not.

OVERVIEW OF THE REPORT

The document is structured as follows:

Chapter 1 gives the introduction, project description, and project objectives. Chapter 2 gives motivation for the work. Chapter 3 gives background information and a brief introduction to some related work.Chapter 4 gives information about the system design and the architecture used. Chapter 5 gives information about the implementation of the system, also gives information about the software used in this project and the issues about the implementation of the system.
Chapter 6 gives information about the evaluation process of the system. Chapter 7 gives my contribution.

MOTIVATION FOR THE WORK

This project has been developed to get people interest in computer vision and face recognition. Through the development of techniques like neural network computers can now outperform humans in many face recognition task, particularly those in which large database of faces must be searched.

WHY THIS WORK IS DONE

Face recognition has recently received a blooming attention and interest from the scientific community as well as from the general public. The interest from the general public is mostly due to the recent events of terror around the world, which has increased the demand for useful security systems. This work has been done to find out the various problems relating to face recognition and possible solution to these problems.

PROBLEMS ENCOUNTERED

There are several problems encountered in the development of this project. This includes the following:

The first problem encountered was the image data to be used in developing this project. It was observed that 16 of the 640 images taken have glitches due to problems with the camera setup. Some people had more glitches than others. These bad images have been shown below:

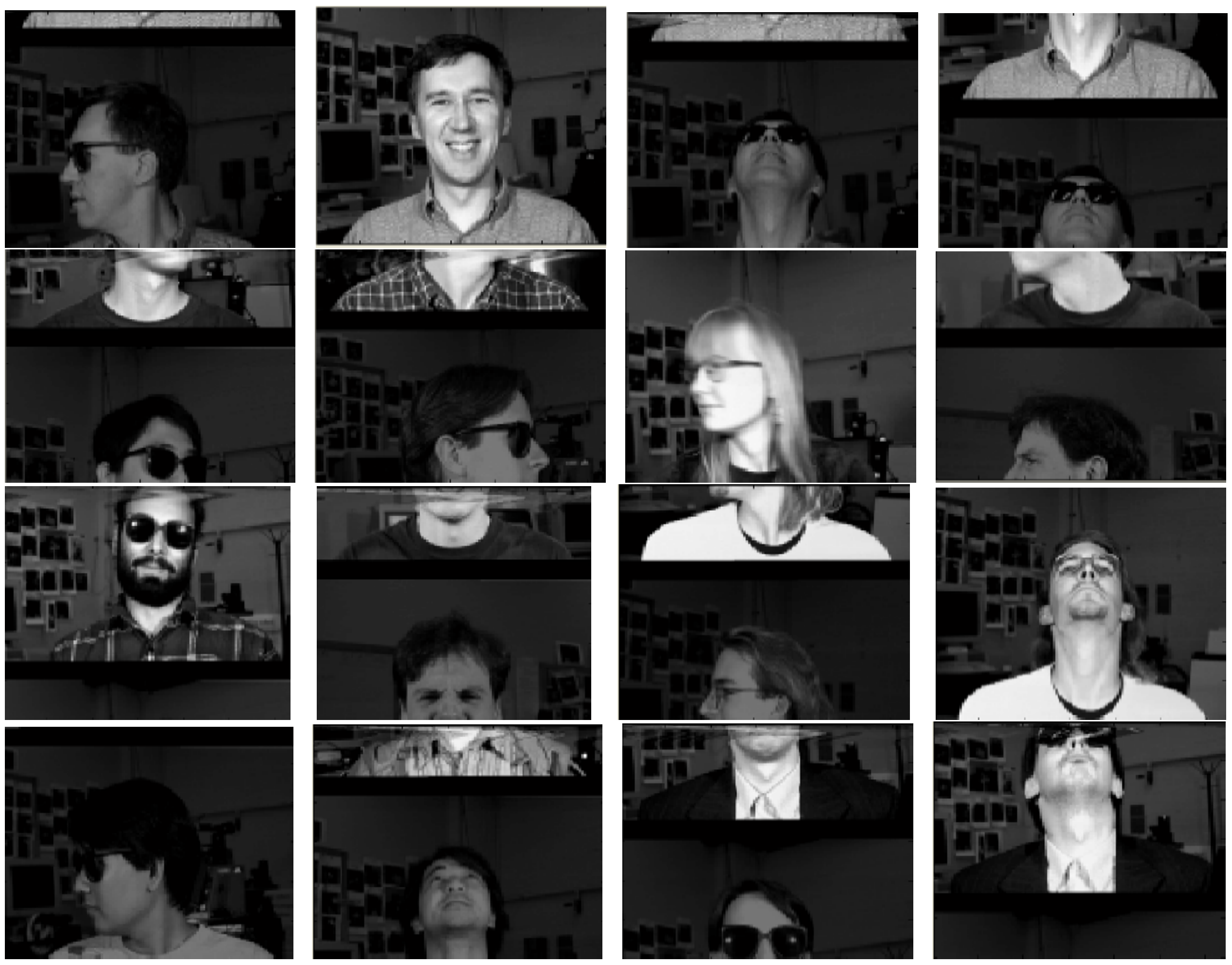


FIGURE 1: BAD IMAGES

The second problem encountered was the programming language to use to get the work done.

The third problem encountered was the data size to be used to be able to achieve the aim of this project.

The overall problem is to be able to design neural network capable of recognizing a person´s identity, pose, and expression.

PROPOSED SOLUTIONS

The problem with glitches on pictures where solved by ignoring those pictures. To reduce the data size only pictures with the smallest resolution (30 x 32) where taken into account.

BACKGROUND READING AND RELATED WORK

The most intuitive way to carry out face recognition is to look at the major features of the face and compare these to the same features on other faces. Some of the earliest studies on face recognition were done by Bledsoe [1966a] was the first to attempt semi-automated face recognition with a hybrid human-computer system that classified faces on the basis of fiducially marks entered on photographs by hand. Parameters for the classification were normalized distances and ratios among points such as eye corners, mouth corners, nose tip, and chin point. Later work at Bell Labs developed a vector of up to 21 features, and recognized faces using standard pattern classification techniques.

Fischer and Elschlager, attempted to measure similar features automatically. They described a linear embedding algorithm that used local feature template matching and a global measure of fit to find and measure facial features. This template matching approach has been continued and improved by the recent work of Yuille and Cohen. Their strategy is based on deformable templates, which are parameterized models of the face and its features in which the parameter values are determined by interactions with the face image.

Connectionist approaches to face identification seek to capture the configurationally nature of the task. Kohonen and Kononen and Lehtio describe an associative network with a simple learning algorithm that can recognize face images and recall a face image from an incomplete or noisy version input to the network. Fleming and Cottrell extend these ideas using nonlinear units, training the system by back propagation.

Others have approached automated face recognition by characterizing a face by a set of geometric parameters and performing pattern recognition based on the parameters. Kanade's face identification system was the first system in which all steps of the recognition process were automated, using a top-down control strategy directed by a generic model of expected feature characteristics. His system calculated a set of facial parameters from a single face image and used a pattern classification technique to match the face from a known set, a purely statistical approach depending primarily on local histogram analysis and absolute gray-scale values.

Recent work by Burt uses a smart sensing approach based on multiresolution template matching. This coarse to fine strategy uses a special purpose computer built to calculate multiresolution pyramid images quickly, and has been demonstrated identifying people in near real time.

NEURAL NETWORKS

An artificial neural network (ANN) is an information processing system that has certain performance characteristics in common with a biological neural network. Neural networks are composed by a large number of elements called *neurons* and provide practical methods for learning real valued, discrete-valued, and vector-valued target functions. There are several network architectures; the most commonly used are: feed forward networks and recurrent networks.

Given network architecture the next step is the training of the artificial neural network (ANN). One learning algorithm very commonly used is back-propagation. This algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It uses gradient descent to minimize the squared error between the network output values and the target values.

APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

Artificial neural network (ANN) learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as input from cameras and microphones. It is also applicable to problems for which more symbolic representations are often used, such as the decision tree learning tasks. In these cases ANN and decision tree learning often produce results of comparable accuracy (Mitchell, 1997).

The back-propagation algorithm is the most commonly used ANN learning technique. It is appropriate for problem with the following characteristics (Mitchell, 1997):

Instances are represented by many attribute value pairs. The target function to be learned is defined over instances that can be described by a vector of predefined features, such as the pixel values. These input attributes may be highly correlated or independent of one another. Input values can be any real values.

The target function output may be discrete-valued, real-valued, or a vector of several real – or discrete-valued attributes.

The training examples may contain errors. ANN learning methods are quite robust to noise in the training data.

Long training times are acceptable. Network training algorithms typically require longer training times than, say, decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the setting s of various learning algorithm parameters.

Fast evaluation of the learning target function may be required. Although ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast.

The ability of humans to understand the learned target function is not important. The weights learned by neural networks are often difficult for humans to interpret. Learned neural networks are easily communicated to humans than learned rules.

APPLICATIONS

The 1988 DARPA Neural Network Study (1) lists various neural network applications, beginning with the adaptive channel equalizer in about 1984. This device, which is an outstanding commercial success, is single-neuron network used in long distance telephone systems to stabilize voice signals. The DARPA report goes on to list other commercial applications, including a small word recognizer, a process monitor a sonar classifier and a risk analysis system. Neural networks have been applied in many fields since the DARPA report was written.

SYSTEM DESIGN

DATA CHARACTERISTICS

The task here involves classifying camera images of faces of people in various pose. Images of 20 different people were collected, including approximately 32 images per person, varying the person's identity, pose, expression, eyes, and size.

<Userid> is the user id of the person in the image and this filed has 20 values shown below.

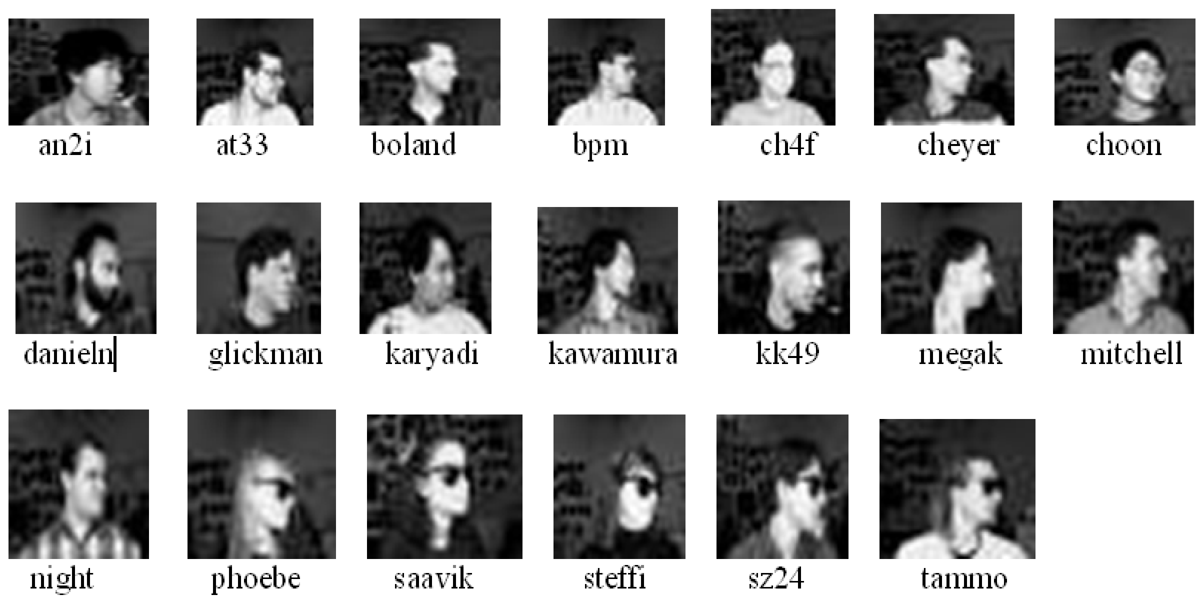


FIGURE 2: USERID IMAGES

<Pose> is the head position of the person, and this field has 4 values: right, left, straight, up

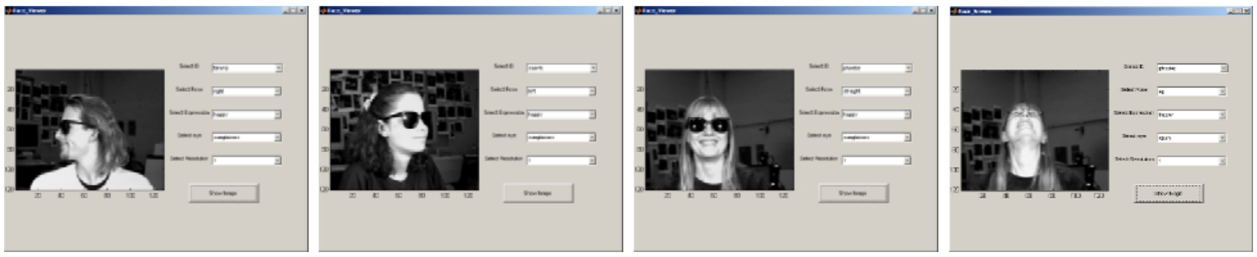
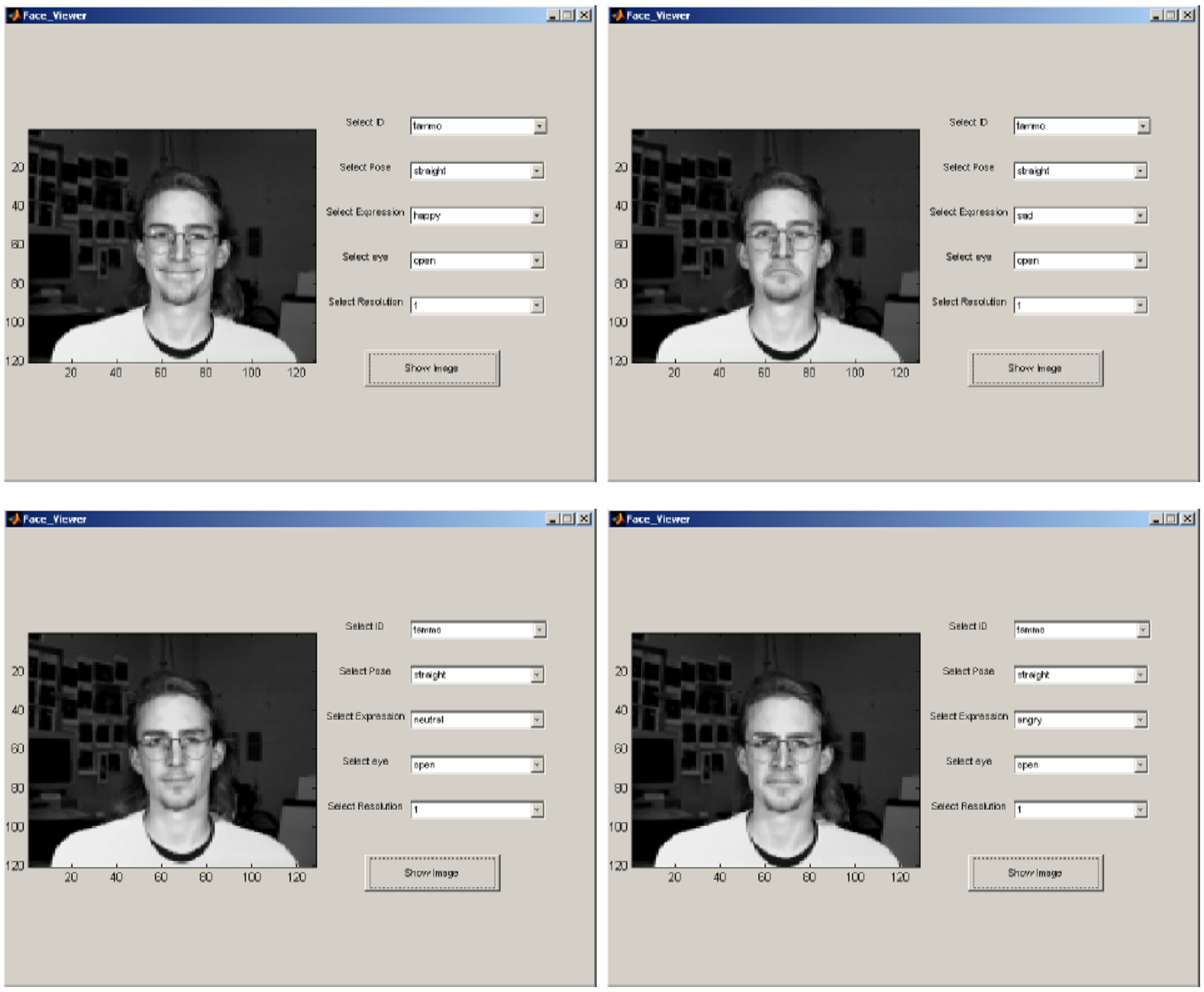


FIGURE 3: POSE IMAGES

<Expression> is the facial expression of the person, and this has 4 values: happy, sad, neutral, sad



Select ID: tammo
Select Pose: straight
Select Expression: happy
Select eye: open
Select Resolution: 1

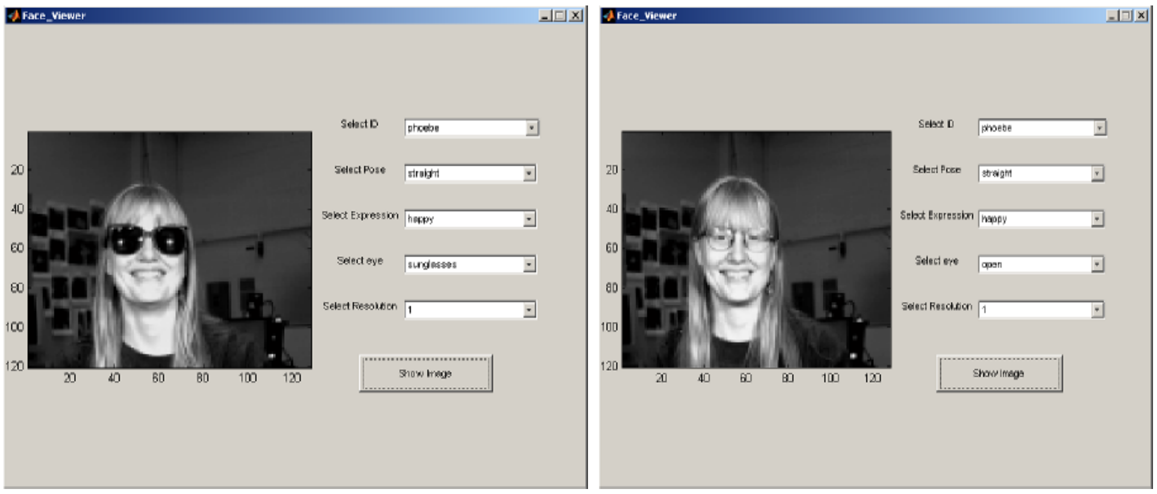
Select ID: tammo
Select Pose: straight
Select Expression: sad
Select eye: open
Select Resolution: 1

Select ID: tammo
Select Pose: straight
Select Expression: neutral
Select eye: open
Select Resolution: 1

Select ID: tammo
Select Pose: straight
Select Expression: angry
Select eye: open
Select Resolution: 1

FIGURE 4: EXPRESSION IMAGES

<Eyes> is the eye state of the person and this field has 2 values: sunglasses and open

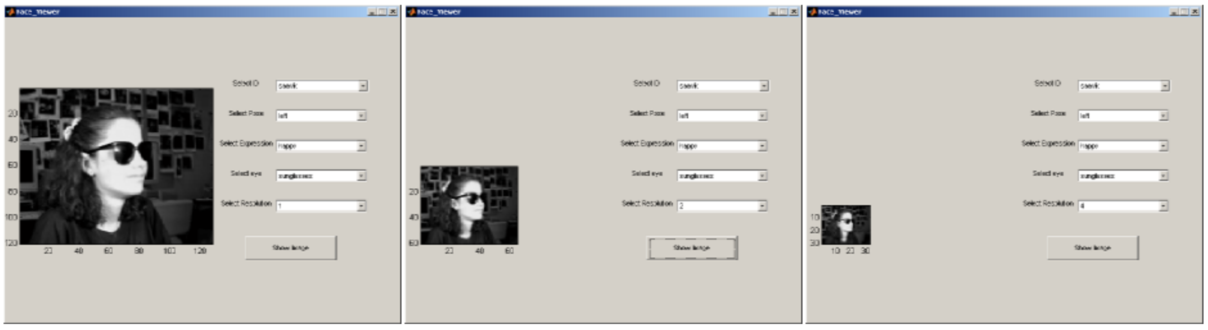


Select ID: phoebe
Select Pose: straight
Select Expression: happy
Select eye: sunglasses
Select Resolution: 1

Select ID: phoebe
Select Pose: straight
Select Expression: happy
Select eye: open
Select Resolution: 1

FIGURE 5: EYE IMAGES

<Size> is the scale of the image and this filed has 3 values: 1, 2, and 4. 1 indicates a full resolution image (128 columns by 120 rows); 2 indicates a half resolution (64 x 60); 4 indicates a quarter resolution image (32 x 30).



Select ID: saavik
Select Pose: left
Select Expression: happy
Select eye: sunglasses
Select Resolution: 1

Select ID: saavik
Select Pose: left
Select Expression: happy
Select eye: sunglasses
Select Resolution: 2

Select ID: saavik
Select Pose: left
Select Expression: happy
Select eye: sunglasses
Select Resolution: 4

FIGURE 6: RESOLUTION IMAGES

REQUIREMENTS

FUNCTIONAL REQUIREMENTS

- The software should be able to recognize a sample face from a set of given faces
- Given some representation of image, the software should be able to preprocess the image and then input these features into the network.

- The artificial neural network should identify the direction in which the person is looking (left, right, up, or straight).
- The artificial neural network should identify whether the person is wearing sunglasses
- The artificial neural network should identify the id of the person
- Use a simple approach for recognition and training.

NON-FUNCTIONAL REQUIREMENTS

This project is intended to meet the following non functional requirements:

- The software would be available at University library, to enable the users to use, at any time.
- The program should be platform independent.

NEURAL NETWORK

An artificial neural network (ANN) is an information processing system that has certain performance characteristics in common with a biological neural network. Neural networks are composed by a large number of elements called *neurons* and provide practical methods for learning real valued, discrete-valued, and vector-valued target functions. There are several network architectures; the most commonly used are: feed forward networks and recurrent networks.

NEURONS

A neural network consists of layers of interconnected “artificial neurons”, as shown in Figure 7 below “neuron” in a neural network is sometimes called a “node” or “unit.

WHY NEURAL NETWORK

Neural networks can be trained to solve problems that are difficult for conventional computers or human beings.

Neural networks can also been trained to perform complex functions in various fields, including pattern recognition, identification, classification, speech, vision, and control systems.

NETWORK GRAPH STRUCTURE

A multilayer feedforward neural network consists of a layer of input units, one or more layers of hidden units, and one output layer of units. A neural network that has no hidden units is called a Perceptron. However, a perceptron can only represent linear functions, so it is not powerful enough for the kind of application I want to solve. On the other hand, a multilayer feed-forward neural network can represent a very broad set of nonlinear functions. So, it is very useful in practice.
For the purpose of this project, I will concentrate on the multilayer feed-forward neural networks which can express most nonlinear functions.

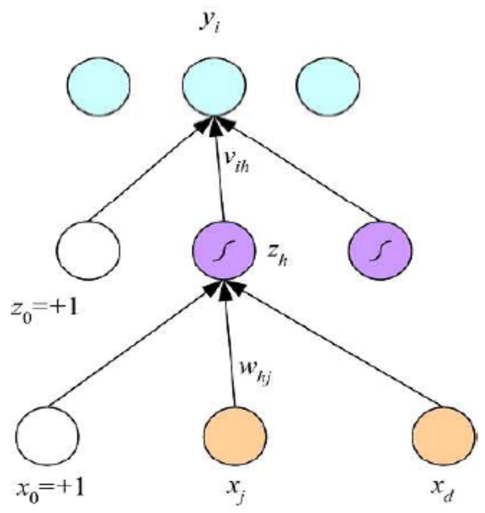


FIGURE 7: STRUCTURE OF MULTILAYER FEEDFOWARD ARTIFICIAL NEURAL NETWORK

Here, there is one layer of input nodes (shown in the bottom row), one layer of hidden nodes (i.e., the middle row), and one layer of output nodes (at the top). The number of nodes per layer is application-dependent.

This structure is called multilayer because it has a layer of processing units (i.e., the hidden units) in addition to the output units. These networks are called feedforward because the output from one layer of neurons feeds forward into the next layer of neurons. There are never any backward connections, and connections never skip a layer. Typically, the layers are fully connected, meaning that all units at one layer are connected with all units at the next layer. So, this means that all input units are connected to all the units in the layer of hidden units, and all the units in the hidden layer are connected to all the output units. Usually, determining the number of input units and output units is clear from the application.

However, determining the number of hidden units is a bit of an art form, and requires experimentation to determine the best number of hidden units. Too few hidden units will prevent the network from being able to learn the required function, because it will have too few degrees of freedom. Too many hidden units may cause the network to tend to over fit the training data, thus reducing generalization accuracy. In many applications, some minimum number of hidden units is needed to learn the target function accurately, but extra hidden units above this number do not significantly affect the generalization accuracy, as long as cross validation techniques are used. Too many hidden units can also significantly increase the training time. Each connection between nodes has a *weight* associated with it. In addition, there is a special weight (called w_0) that feeds into every node at the hidden layer and a special weight (called z_0) that feeds into every node at the output layer. These weights are called the bias, and set the thresholding values for the nodes. Initially, all of the weights are set to some small random values near zero. The training of the network will adjust these weights using the backpropagation algorithm that so that the output generated by the network matches the correct output.

PROCESSING THE NODE

Every node in the hidden layer and in the output layer processes its weighted input to produce an output. This can be done slightly differently at the hidden layer, compared to the output layer. Here is shown how it works.

INPUT ENCODING

The input data we provide to the network comes through the inputs units. No processing takes place in an input unit. It simply feeds data into the system. For example, if we are inputting a grayscale image into the network, the picture will be divided into pixels (say, 120 x 128 pixels), each of which is presented by a number typically in the range from 0 to 255 that says what the grayscale value is for that piece of the image. One pixel 0 to 255 will be fed into each input unit. So if you have an image of 120 x 128 pixels you will have 15360 input units. The value coming out of an input is labeled x_j, for j going from 1 to d, representing d input units. There is also a special input unit labeled x_0, which always has the value of 1. This is used to provide the bias to the hidden nodes.

HIDDEN UNITS

The connections coming out of an input unit have weights associated with them. A weight going to hidden unit z_h from input unit x_j would be labeled wh_j. The bias input node, x_0, has a weight of w_0. In the training, this bias weight, w_0, is treated like all other weights, and is updated according to the backpropagation algorithm; the value coming out of x_0 is always 1.

Each hidden node calculates the weighted sum of its inputs and applies a thresholding function to determine the output of the hidden node. The weighted sum of the inputs for hidden node z_h is calculated as:

$$\sum_{j=0}^{d} w_{hj} x_j$$

EQUATION 1: WEIGHTED SUM OF INPUTS

The thresholding function applied at the hidden node is typically either a step function or a sigmoid function. For the purpose of this project, I will stick with the sigmoid function. The general form of the sigmoid function is:

$$\mathrm{sigmoid}(a) = \frac{1}{1+e^{-a}}$$

EQUATION 2:SIGMOID FUNCTION

Often, we preprocess the input to reduce the number of input units in the network. So, in this case, we might average several pixels to reduce the resolution down to, say, 30 x 32, which are just 960 input units.

The sigmoid function is sometimes called the "squashing" function, because it squashes its input to a value between 0 and 1. At the hidden node, we apply the sigmoid function to the weighted sum of the inputs to the hidden node, so we get the output of hidden node z_h is:

$$z_h = \mathrm{sigmoid}\left(\sum_{j=0}^{d} w_{hj}x_j\right) = \frac{1}{1+e^{-\sum_{j=0}^{d} w_{hj}x_j}}$$

EQUATION 3:OUTPUT OF HIDDEN NODE

for *h* going from *H*, where *H* is the total number of hidden nodes.

OUTPUT ENCODING

Now, we can do a similar computation for the output nodes. The computation also depends on whether we have 1 output unit or multiple output units.

We start out in the same way as we did with the hidden units, calculating the weighted sum. We label the weights going into output unit *i* from hidden unit *h* as v_{ih}. Just like the input layer, we also have a bias at the hidden layer. So, each output unit has a bias input from hidden unit z_0, where the input from z_0 is always 1 and the weight associated with that input is trained just like all the other weights.
So, output unit *i* computes the weighted sum of its inputs as:

$$o_i = \sum_{h=0}^{H} v_{ih} z_h$$

EQUATION 4:WEIGHTED SUM OF ITS INPUTS

If there is just one output unit then, we omit the *i* subscripts and we have:

$$o = \sum_{h=0}^{H} v_h z_h$$

EQUATION 5:WEIGHTED SUM OF ITS INPUTS 2

Now, we have to decide what function we are going to apply to this weighted sum to generate y_i, which is the output of unit i.

TRAINING THE NETWORK

TRAINING

Training the neural network to produce the correct outputs for the given inputs is an iterative process, in which you repeatedly present the network with an example, compare the output on this example, sometimes called the *actual output* with the desired output sometimes called the *target output*, and adjust the weights in the network to hopefully generate better output. By training the network over and over with various examples, and using the backpropagation algorithm to adjust the weights, the network should learn to produce better result. Ideally, the "better result" is not just the right answer for the data that you train your network on, but also for generalizations of that data.

We can train the network using a data set of examples (called *training data*). For each example, you know the correct answer, and you tell the network this is correct answer. We will call the process of running 1 example through the network and training the network on that example, *weight update iteration*. Training the network once on each example of the training set is called an *epoch*. Typically, we have to train your network for many epochs before it *converges*, meaning that the network has settled in on a function that it thinks is the best predictor of your input data.

BACKPROPAGATION ALGORITHM

The algorithm used to train the network is the backpropagation Algorithm. The general idea with the backpropagation algorithm is to use gradient descent to update the weights so as to minimize the squared error between the network output values and the target output values. The update rules are derived by taking the partial derivative of the error function with respect to the weights to determine each weight's contribution to the error. Then, each weight is adjusted, using gradient descent, according to its contribution to the error.

This process occurs iteratively for each layer of the network, starting with the last set of weights, and working back towards the input layer, hence the name backpropagation.

The backpropagation algorithm is the most common network learning method and has been successfully applied to a variety of tasks, such as handwriting recognition and robot control. The hypothesis space considered by the back-propagation algorithm is the space of all functions that can be represented by assigning weights to the given, fixed network of interconnected units. Feedforward networks containing three layers of units are able to approximate any function to arbitrary accuracy, given a sufficient (potentially very large) number of units in each layer. Even networks of practical size are capable of representing a rich space of highly nonlinear functions, making feedforward networks a good choice for learning discrete and continuous functions whose general form is unknown in advance. (Mitchell, 1997, p. 122).

Back-propagation searches the space of possible hypothesis using gradient descent to iteratively reduce the error in the network fit to the training example. Gradient descent converges to a local minimum in the training error with respect to the network weights. More generally, gradient descent is potentially useful method for searching many continuously parameterized hypothesis spaces where the training error is a differentiable function of hypothesis parameters. (Mitchell, 1997, p. 123).

One of the most intriguing properties of back-propagation is its ability to invent new features that are not explicit in the input to the network. In particular, the internal (hidden) layers of multilayer networks learn to represent intermediate features that are useful for learning the target function and that are only implicit in the network inputs. (Mitchell, 1997, p. 123).

Although back-propagation is the most common Artificial Neural Network (ANN) learning algorithm, many other have been proposed, including algorithms for more specialized tasks. For example, recurrent neural network methods train networks containing directed cycles; algorithms such as CASCADE CORRELATION alter the network structure as well as the network weights. (Mitchell, 1997, p. 123).

ONLINE LEARNING

Offline learning occurs when we compute the weight updates after summing over all of the training examples.

ONLINE UPDATE

Online learning occurs when we update the weights after each training example. The theoretical difference between the two approaches is that offline learning implements what is called *Gradient Descent*, whereas online learning implements *Stochastic Gradient Descent*. The general approach for the weight updates is the same, whether online or offline learning is used. The only difference is that offline learning will sum the error over all inputs, while the online learning will compute the error for each input one at a time.

OTHER LEARNING ALGORITHM PARAMETERS

MOMENTUM

As the learning is generally slow, momentum was used to increase the performance. In this method the weight update of the previous iteration are taken into account, by adding α (momentum) times the previous weight update to the current one, this can be seen on the following equation:

$$w_{hj}^{t} = w_{hj}^{t} + \Delta w_{hj}^{t} + \alpha \Delta w_{hj}^{t-1}$$

EQUATION 6: MOMENTUM

LEARNING RATE

In these weight updates, we also use a positive constant learning rate, that moderates the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.3), and is sometimes made to decay as the number of weight-tuning iterations increases.

NETWORK OVERFIT

The best way to determine whether your network has reached the best set of weights for the training data is to validate the results using a validation set of data. This is a separate data set that you do not use during training. Instead, you use the validation data to detect when your network is beginning to overfit to the training data. We dont want the network to generalize its inputs, so that it can correctly answer classification queries not only for the training data, but also for other examples. If we train the network too long, it will overfit to the training data, which means that it will correctly answer only examples that are in the training data set.

To help ensure that the network does not overfit, we can use a cross validation procedure.

CROSS VALIDATION

In cross validation the input set is split into a separate training and a validation set. The network is trained for several iterations and the error calculated. Afterwards the error in applying the network to the validation set is calculated as well. This training and validation process is repeated until the validation error begins to increase, i.e. overfitting takes place. The training should be stopped then as the network approaches the patterns of the training set rather than the ones of the input set. Training can be also stopped if there is no significant increase in accuracy any more.

In case of 10-fold-cross validation the dataset is split into 10 sets. For each of these sets the network is trained with the other 9 sets and validated with the one excluded. Error an accuracy after a certain number of iterations is recorded. Finnally the average of errors and accuracies is calculated.

SYSTEM IMPLEMENTATION

SOFTWARE USED

Matlab is an interpreted language (and as such can be much slower than compiled software) for numeric computation and visualization. It offers high level facilities for dealing directly with mathematical constructs.

BENEFIT OF USING MATLAB

Benefits that it offers for this project are:

1. Excellent support for linear algebra and matrix operations. The basic type in Matlab is a double precision matrix. The software was originally developed as a linear algebra package (Matlab stands for MATrix LABoratory) and has efficient and numerically reliable algorithms for matrix inversion, eigenvalues etc.

2. Visualization facilities. The in-built graphing and plotting functions are easy to use for both 2d and 3d plots.

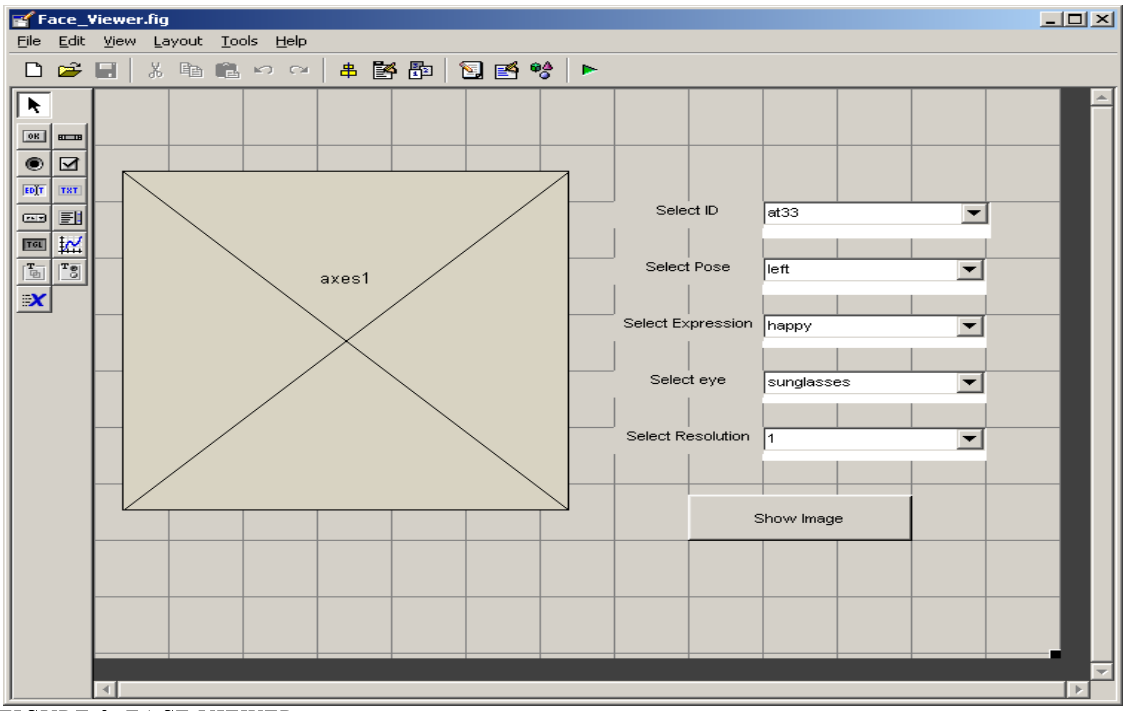
3. Ease of extension. Functions and scripts can be written in the Matlab language (in `M-files') and these can then be called in exactly the same way as the core functionality of Matlab. In fact, the `toolboxes' that extend the functionality of Matlab to more specialist areas are written in this way. Netlab (a Matlab toolbox for neural networks) consists of a set of M-files.

4. Portability. Software written in the Matlab language is portable to any platform that runs Matlab, including UNIX machines, PCs and Macintoshes.

IMPLEMENTATION ISSUES

GRAPHICAL USER INTERFACE

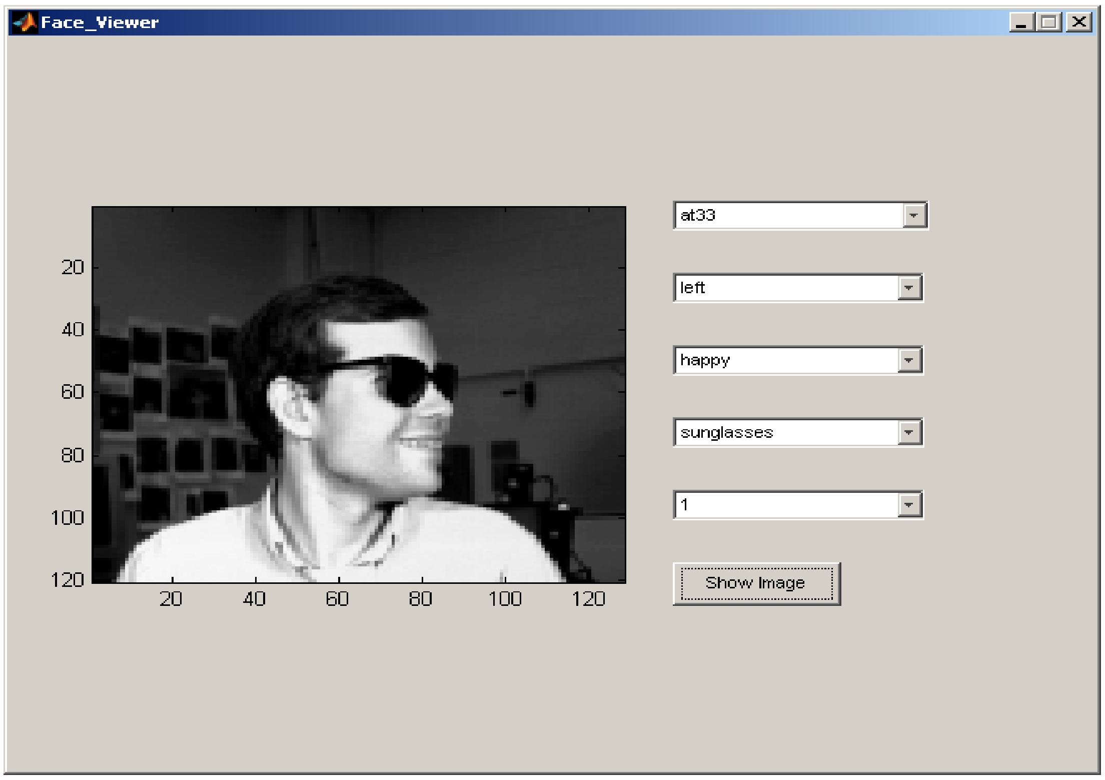
The Graphical User Interface was constructed using *MatLab GUIDE* or *Graphical User Interface Design Environment*. Using the layout tools provided by *GUIDE*, I designed the following graphical user interface figure (face_viewer.fig) for the face recognition user application:



Select ID: at33
Select Pose: left
Select Expression: happy
Select eye: sunglasses
Select Resolution: 1

FIGURE 8: FACE VIEWER

The above design yields the following application window on run time:



at33
left
happy
sunglasses
1

FIGURE 9: FACE VIEWER

PREPROCESSING

Several MatLab files were created responsible for extracting the picture information into a MatLab .mat file. The idea is to scan each of the folders containing the images and load each of the pictures into a matrix. The target matrix in dimension *number of cases * total number of pictures* is filled in the following manner: The weight 0.9 is assigned to the actual target class for each picture, while 0.1 is assigned to all other non-target classes. E.g. if a mat-file for the pose recognizer should be created the target classes are (left, right, straight and up). If a picture points left the entry in the target matrix would look like (0.9, 0.1, 0.1, 0.1), further details are outlined in the following table.

Point direction	**Left**	**Right**	**Straight**	**Up**
Left	0.9	0.1	0.1	0.1
Right	0.1	0.9	0.1	0.1
Straight	0.1	0.1	0.9	0.1
Up	0.1	0.1	0.1	0.9

TABLE 1: TARGET MATRIX

EXPERIMENTS

For each experiment (e.g. determine the pose of a person) first the mat file created is loaded into the program. The number of neurons, as well as iterations is decided beforehand. For each iteration the backpropagation algorithms is applied, and after a specified number of epochs the error and the accuracy recorded.

The error was calculated using the sum of squares of error method.

POSTPROCESSING

After retrieving the results from the actual experiment attempts were made to decide the optimal number of weight updates, by determining after how many iterations no significant improvement of the accuracy can be achieved, i.e. when overfitting towards the training data occurs.

SUMMARY

MatLab has been used as programming because of its great support for matrix operations, its portability, visualization features and extendibility.

EVALUATION

SUNGLASS RECOGNIZER

The learning task here involves training a neural net which, when given an image as input, indicates whether the in the image is wearing sunglasses or not.

LEARNING PARAMETERS

This is a three layer fully connected feed-forward neural network, which uses the back propagation network to tune its weights. The code is setup to learn to recognize 2 states the eye can be in: open and sunglasses. Sunglasses recognizer implements a neural net that accepts an image as input, and outputs the state of the eye of the person.

Here a 960 x 2 x 2 network is trained on grey level images, to predict whether a person is wearing sunglass or not. Each pixel gray scale value is given as input to the node. The number of hidden nodes has been zeroed on 2 and the number of output is 2.The network has been trained using a default learning parameter of 0.3 learning rate and momentum respectively.

NETWORK WEIGHT

The network weights are shown after 100 weight tuning iteration. Each output unit (open, sunglass) has four weights, shown by dark (negative).

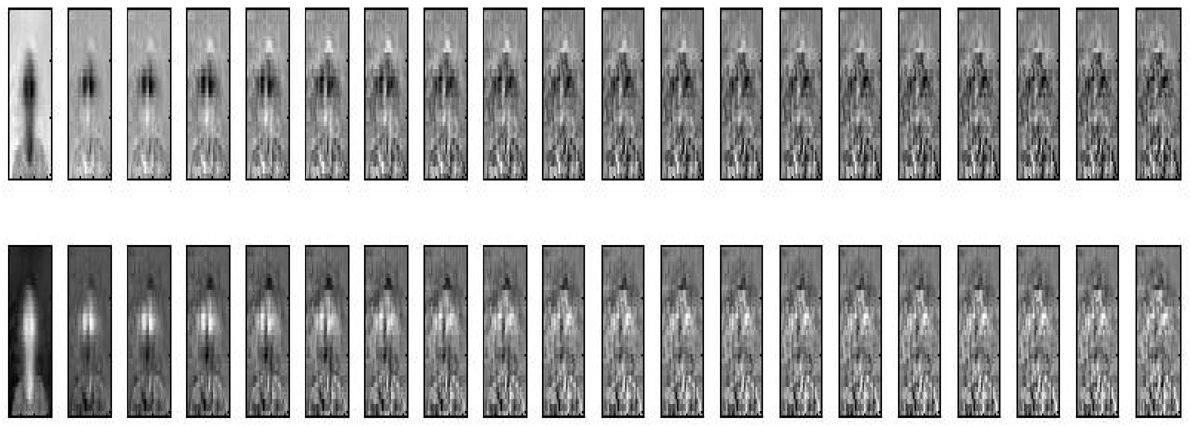


FIGURE 10: NETWEIGHT AFTER 2000 WEIGHT UPDATES

ACCURACY

After training on 642 such images, the network achieves an accuracy of 92% over a separate test set shown below.

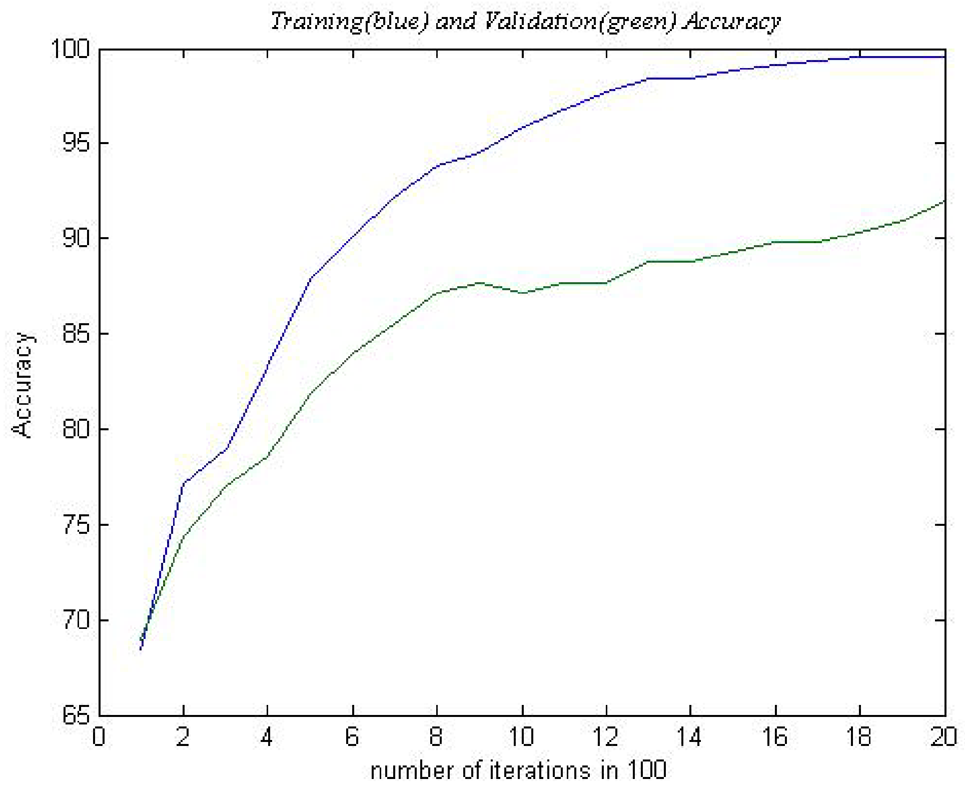


FIGURE 11: NETWORK ACCURACY AFTER 2000 WEIGHT UPDATES

ERROR

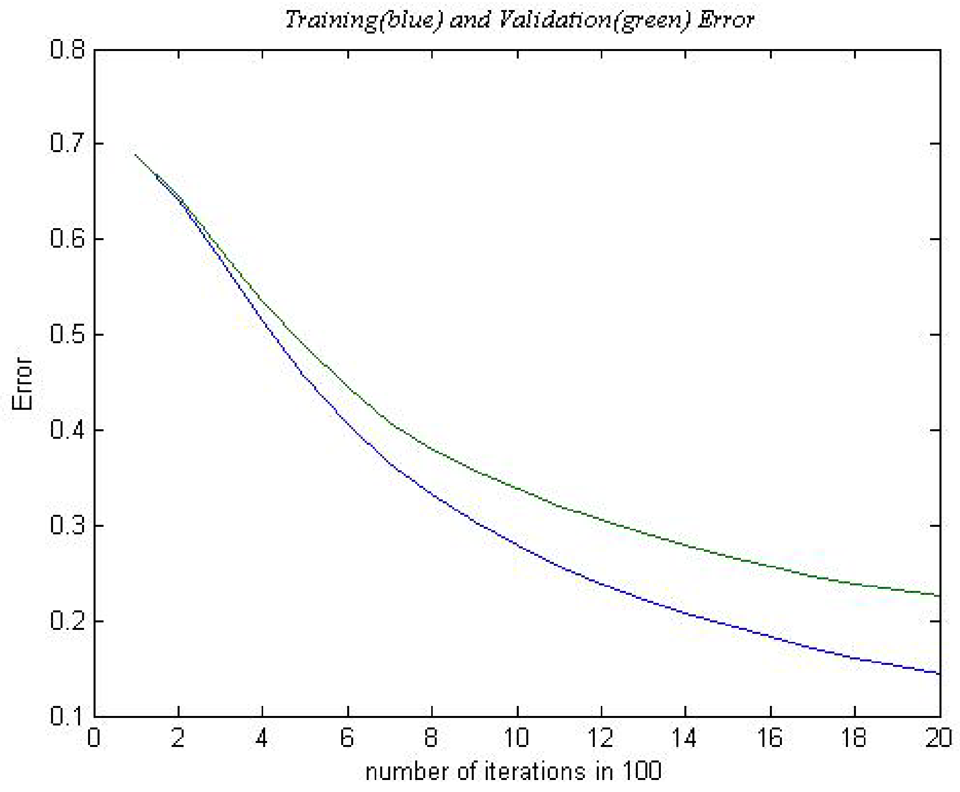


FIGURE 12: NETWORK TRAINING AND VALIDATION ERROR AFTER 2000 WEIGHT UPDATES

FACE RECOGNIZER

The learning task here involves training a neural net that accepts an image as input, and output the userid of the person.

LEARNING PARAMETERS

This is a three layer fully connected feedforward neural network, which uses the back propagation network to tune its weights. The code is setup to learn to recognize who the person in a picture is among a group of 20 possible people.

Here a 960 x 10 x 20 network is trained on grey level images. Each pixel gray scale value is given as input to the node. The number of hidden nodes has been zeroed on 10 and the number of output is 20.The network has been trained using a default learning parameter of 0.3 learning rate and momentum respectively, to recognize who the person in a picture is among a group of 20 possible people.

After training on 642 such images, the network achieves an accuracy of 92% over a separate test set shown below.

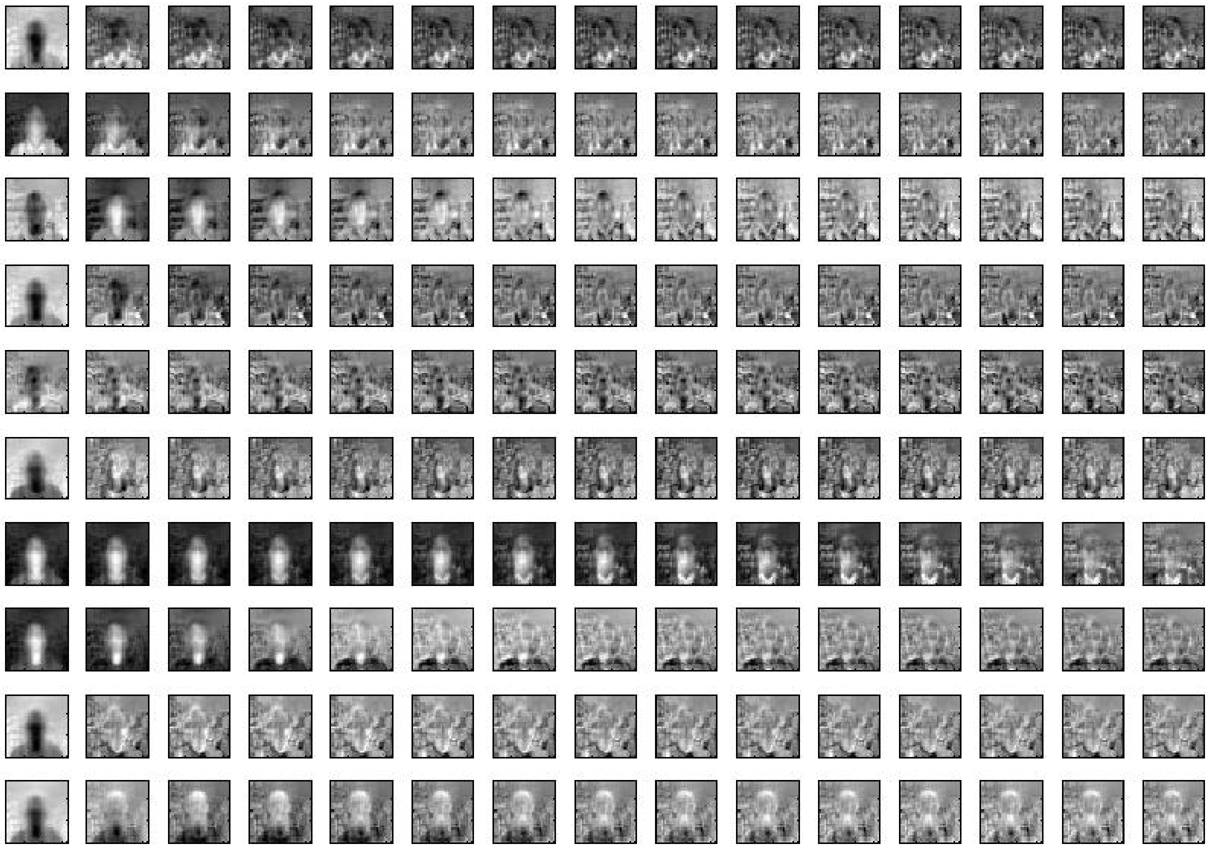


FIGURE 13: NETWORK WEIGHTS AFTER 1500 WEIGHT UPDATES

ACCURACY

After training on 642 such images, the network achieves an accuracy of 95% over a separate test set shown below.

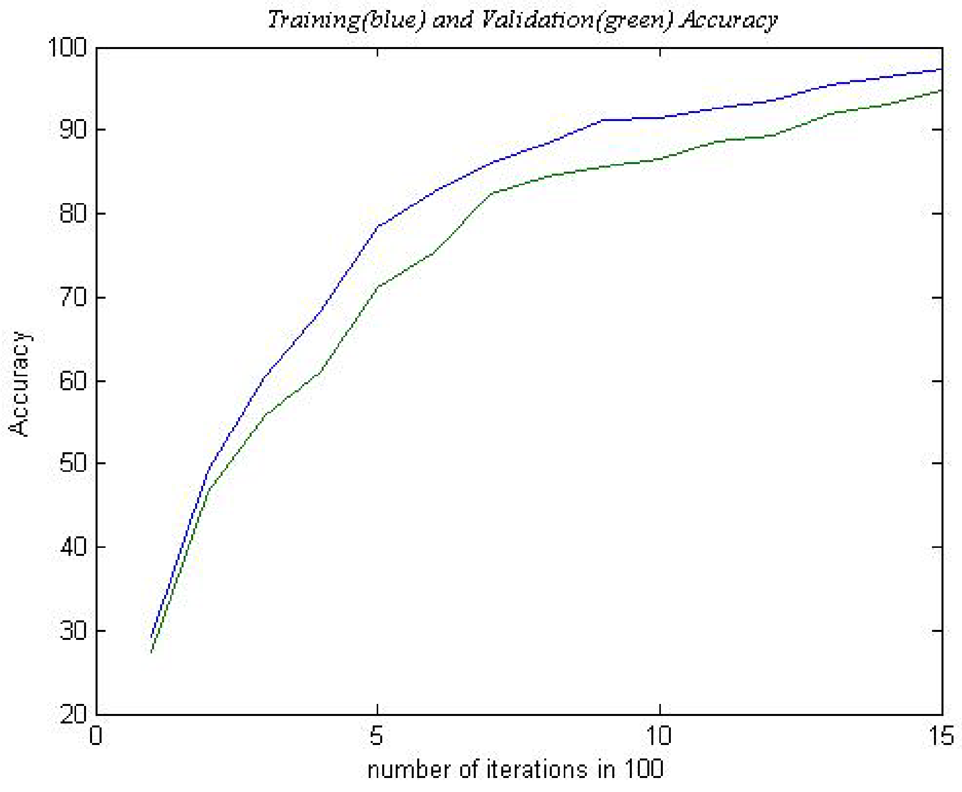


FIGURE 14: NETWORK ACCURACY AFTER 1500 WEIGHT UPDATES

ERROR

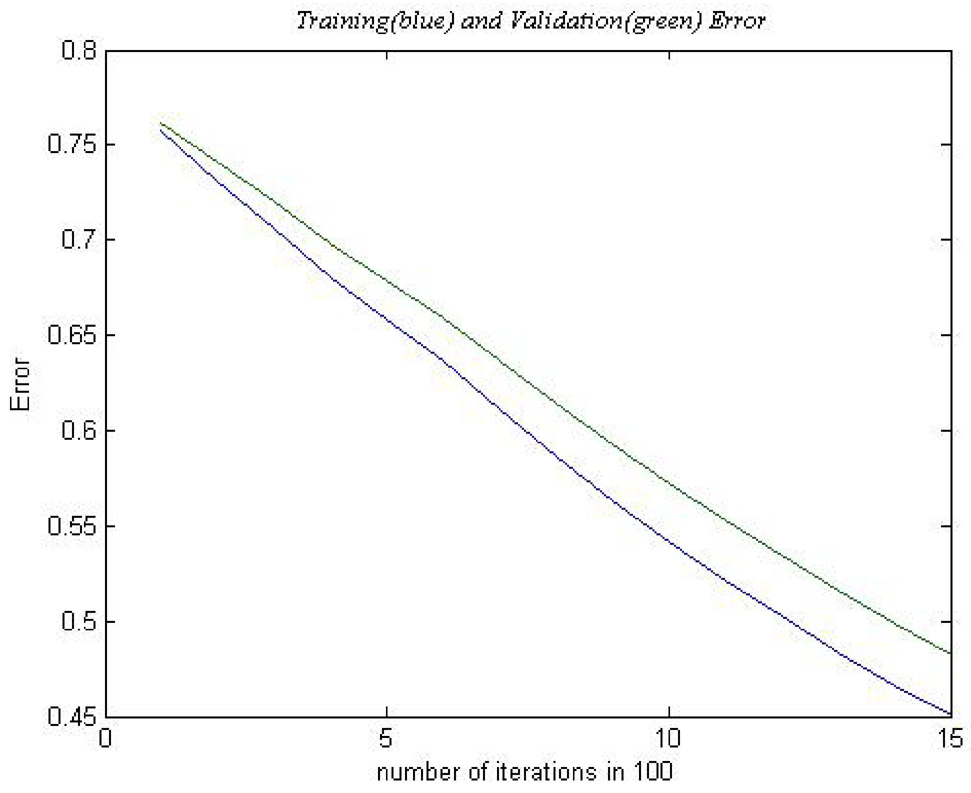


FIGURE 15: NETWORK TRAINING AND VALIDATION ERROR AFTER 1500 WEIGHT UPDATES

POSE RECOGNIZER

This is a three layer fully connected feed-forward neural network, which uses the back propagation network to tune its weights. The code is setup to learn to recognize 4 states left, right, straight and up.. The pose recognizer implements a neural net that accepts an image as input, and outputs the pose of the person.

LEARNING PARAMETERS

Here a 960 x 3 x 4 network is trained on grey level images, to predict whether a person is looking to the left, right straight or up. Each pixel gray scale value is given as input to the node. The number of hidden nodes has been zeroed on 3 and the number of output is 4.The network has been trained using a default learning parameter of 0.3 learning rate and momentum respectively.

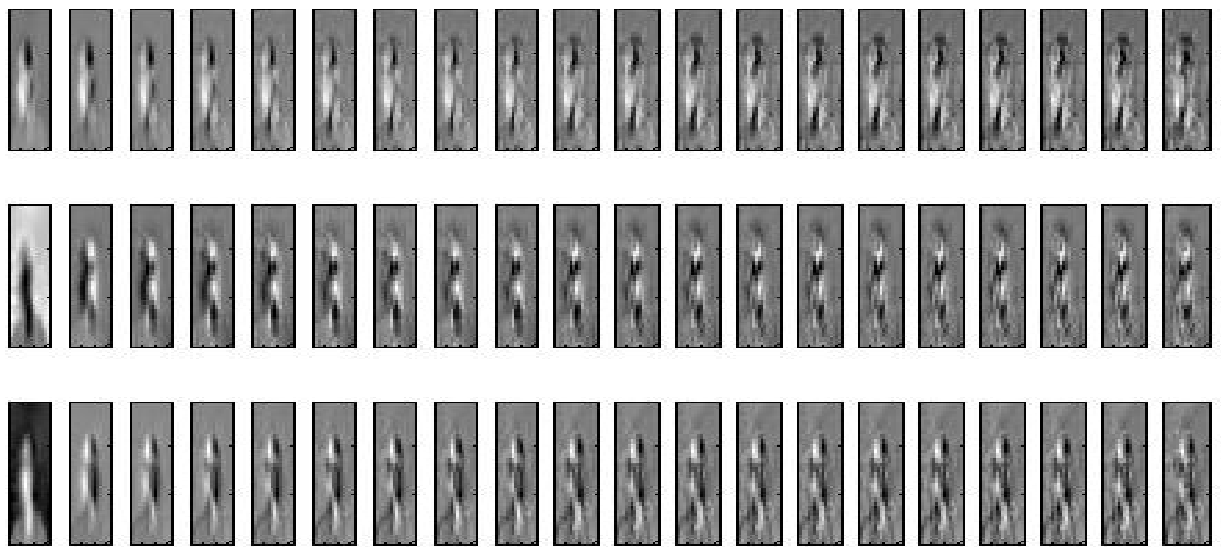


FIGURE 16: NETWORK WEIGHTS AFTER 2000 WEIGHT UPDATES

ACCURACY

After training on 642 such images, the network achieves an accuracy of 92% over a separate test set shown below.

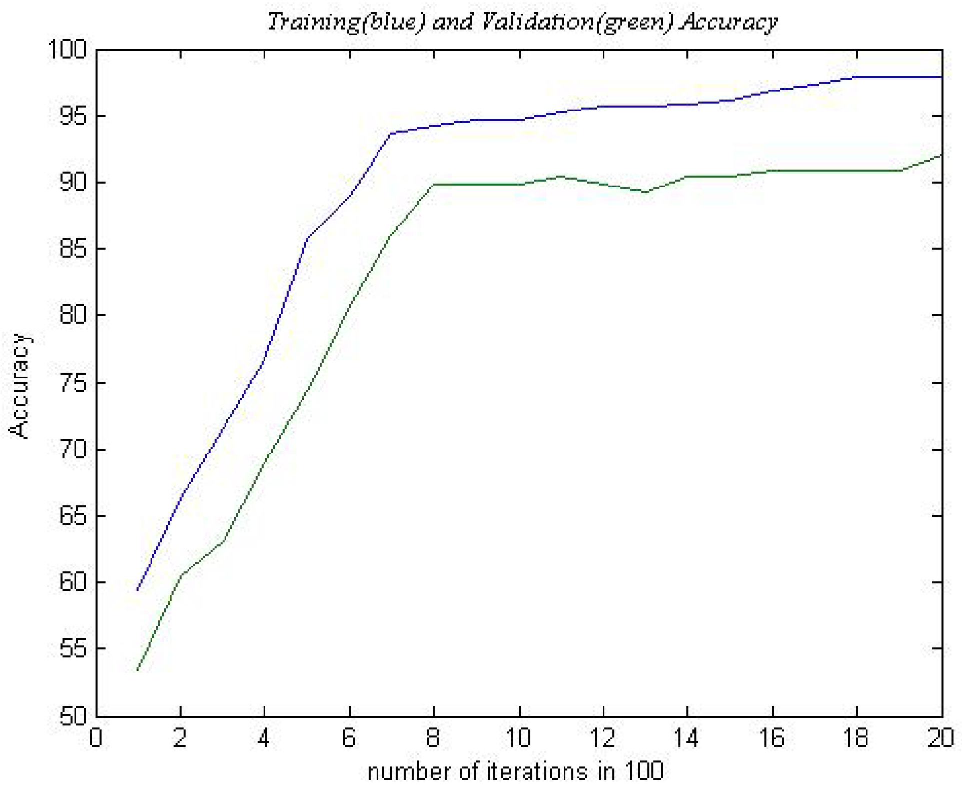


FIGURE 17: NETWORK ACCURACY AFTER 2000 WEIGHT UPDATES

ERROR

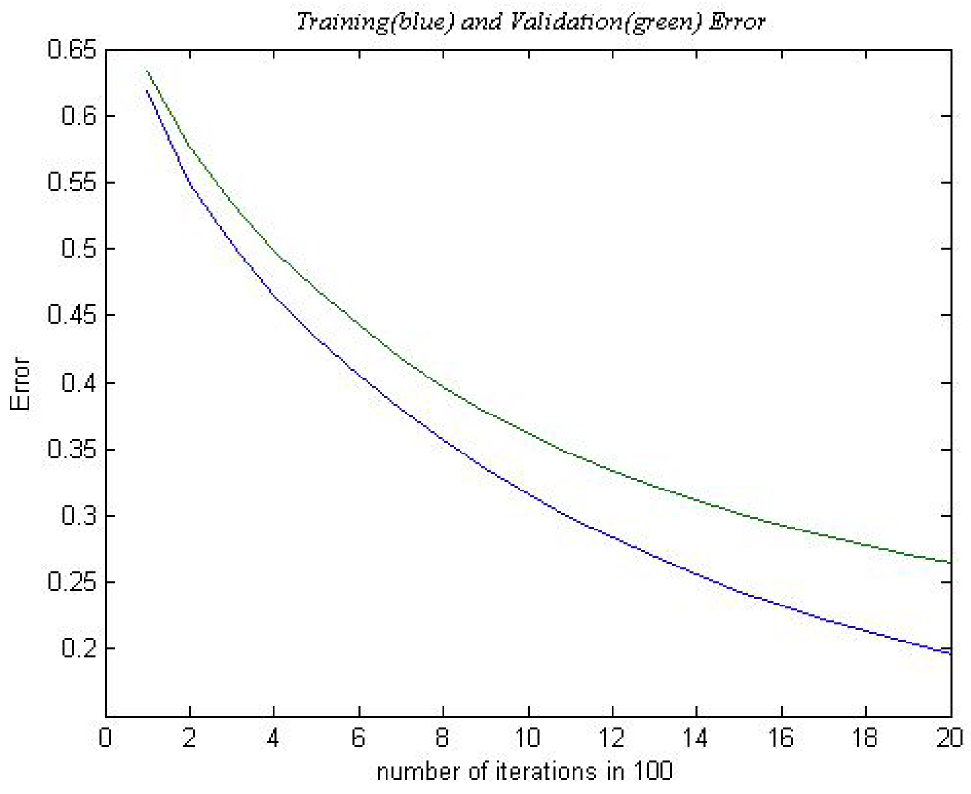


FIGURE 18: NETWORK TRAINING AND VALIDATION ERROR AFTER 2000 WEIGHT UPDATES

(OTHER OUTPUT ENCODING (0.9/0.1 OR 1/0))

If instead of 0.9 and 0.1 for the values in the target matrix 1 and 0 are used, the accuracy is generally lower than incase of 0.9 and 0.1. The reason for this is that the sigmoid function cannot handle absolute values like 0 and 1, the weights for certain pixels will grow up to infinity. As 0.9 and 0.1 allow a certain tolerance for the weights the sigmoid function can be used here.

10 FOLD CROSS VALIDATION

The 10-fold cross validation was applied to the pose dataset.

ERROR

The following figure shows overfitting. First the validation error is less than the training error, but after ca. 50 weight updates the validation error decreases less than the training error, i.e. the networks is more concentrated on the patterns of the training set.

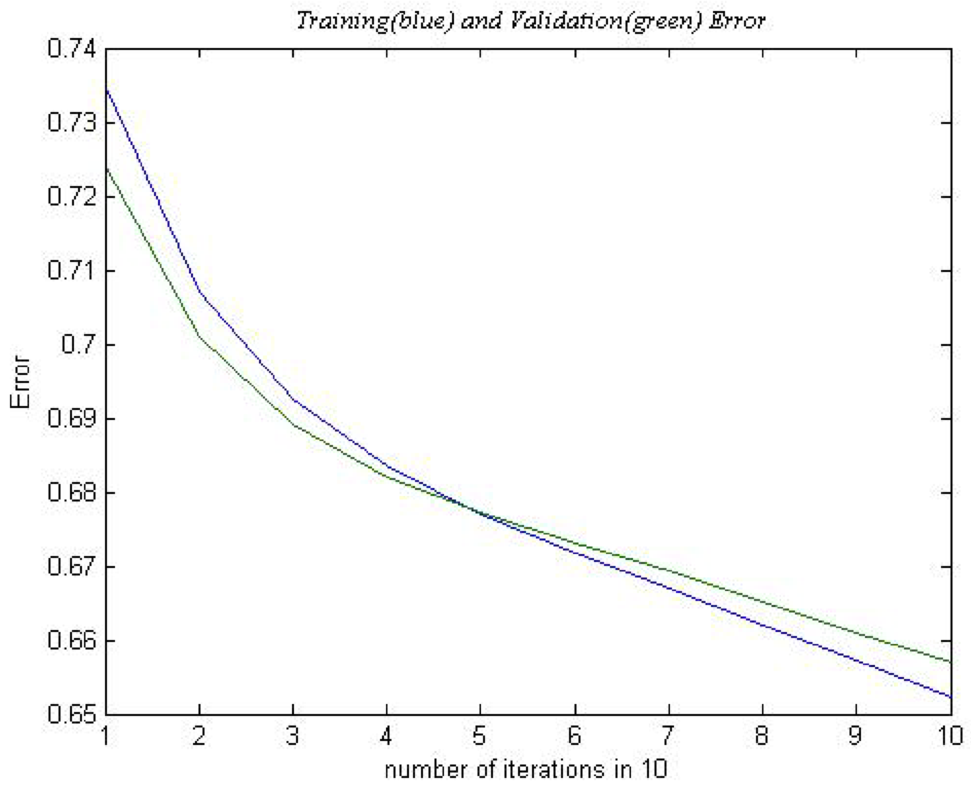


FIGURE 19: NETWORK TRAINING AND VALIDATION ERROR AFTER 100 WEIGHT UPDATES

ACCURACY

After training on 642 such images, the network achieves an accuracy of 53% over a separate test set shown below.

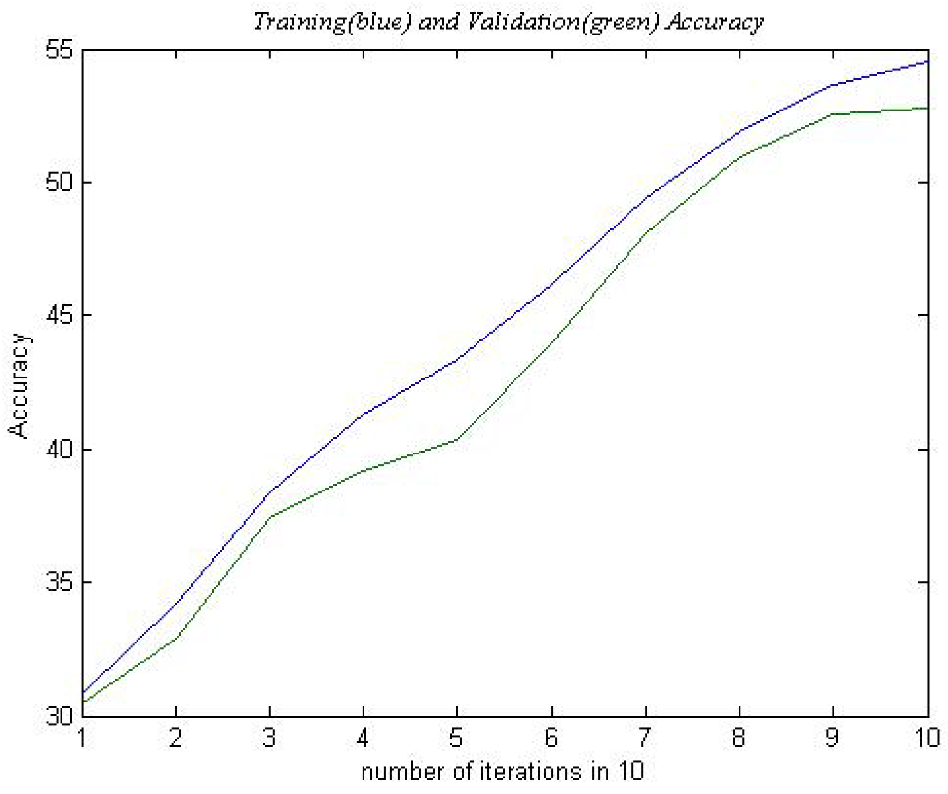


FIGURE 20: ACCURACY AFTER 100 WEIGHT UPDATES

CONCLUSIONS

MY WORK

- learnt how to program in Matlab
- Read a lot on neural network
- design a graphical user interface for the system
- write up matlab code to accomplish the task
- Evaluating accuracy and error result obtained by the training example
- Find out how to tune the network so that it can function better.
- Had discussions with my supervisor.
- Held presentation
- Make user manual for the system
- write the final dissertation

IMPORTANCE AND CONTRIBUTION

The main contribution made to this project is the performance of a 10 fold cross validation experiment using the image data set provided by Dr. Tom Mitchell.

REFLECTION

Working on this project has made me realize how much work in designing and implementing system could be. It has been a good experience.

Working on this project has made me more capable of doing the following.

- writing reports
- Organizing myself
- Organizing my work
- Designing a system
- implementing a system based on designs
- presenting my work
- Avoiding problems.
- Finding resources
- working with neural network
- implementing the system in Matlab
- Working with other people to fulfill the objectives of the project (Dr. Tony Y.T. Chan)

The overall conclusion of this project is that I found, it to be interesting and programming language Matlab and the algorithm interesting subjects to work with in the future.

BIBLIOGRAPHY

1. *Darpa Neural Network Study: October 1987-February 1988.* s.l. : Afcea Intl Pr , 1988.

2. [Online] [Cited: March 23, 2008.] http://www.ai.mit.edu/courses/6.867-f01/matlab.html. .

3. [Online] [Cited: March 3, 2008.] http://www.math.unh.edu/»mathadm/tutorial/software/matlab/.

4. [Online] [Cited: October 2, 2007.] http://www.cs.cmu.edu/~tom/book.html.

5. Neural Network Toolbox™ 6User's Guide. [Online] [Cited: October 15, 2007°.] http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf.

6. Mitchell, Tom M**.** *Machine Learning.* s.l. : McGraw-Hill, 1997.

APPENDIX A: CODE LISTING

FACE_VIEWER.M

```
function varargout = Face_Viewer(varargin)


gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Face_Viewer_OpeningFcn, ...
                   'gui_OutputFcn',  @Face_Viewer_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end


if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end


function Face_Viewer_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
handles.id = 'at33';
handles.pose = 'left';
handles.exp = 'happy';
handles.open = 'sunglasses';
handles.res = '1';
guidata(hObject, handles);



function varargout = Face_Viewer_OutputFcn(hObject, eventdata, handles)
  varargout{1} = handles.output;

function popupmenu1_CreateFcn(hObject, eventdata, handles)
  handles.id_menu = hObject;
  guidata(hObject, handles);
  set(hObject,'BackgroundColor','white');

function popupmenu2_CreateFcn(hObject, eventdata, handles)
  handles.pose_menu = hObject;
  guidata(hObject, handles);
  set(hObject,'BackgroundColor','white');

function popupmenu3_CreateFcn(hObject, eventdata, handles)
  handles.exp_menu = hObject;
  guidata(hObject, handles);
  set(hObject,'BackgroundColor','white');

function popupmenu4_CreateFcn(hObject, eventdata, handles)
```

```
   handles.sun_menu = hObject;
   guidata(hObject, handles);
   set(hObject,'BackgroundColor','white');


function popupmenu5_CreateFcn(hObject, eventdata, handles)
   handles.res_menu = hObject;
   guidata(hObject, handles);
   set(hObject,'BackgroundColor','white');


function pushbutton1_Callback(hObject, eventdata, handles)
   if (strcmp(handles.res,'1') ==1)
      file =strcat('C:\Documents and Settings\Administrator\My Documents\Final Year Project NIK\faces\', handles.id, '\',handles.id,'_',handles.pose,'_',handles.exp, '_', handles.open);
      width = 64;
      height = 20;
   elseif  (strcmp(handles.res,'2') ==1)
      width = 32;
      height = 10;
      file =strcat('C:\Documents and Settings\Administrator\My Documents\Final Year Project NIK\faces\', handles.id, '\',handles.id,'_', handles.pose,'_',handles.exp, '_', handles.open,'_', handles.res);
   else
      width = 16;
      height = 5;
      file =strcat('C:\Documents and Settings\Administrator\My Documents\Final Year Project NIK\faces\', handles.id, '\',handles.id,'_', handles.pose,'_',handles.exp, '_', handles.open,'_', handles.res);
   end;
   position_rectangle = [5, 10, width, height];
    set(gca,'Position',position_rectangle);
    try
       A = imread(file,'pgm');
    catch
       % msgbox('This is one of the error pictures','bad picture','warn')
        file =strcat('C:\Documents and Settings\Administrator\My Documents\Final Year Project NIK\faces\', handles.id, '\',handles.id,'_',handles.pose,'_',handles.exp, '_', handles.open);
        A = imread(strcat(file,'.bad'), 'pgm');
    end
   colormap(gray(256));
   image(A);
function edit1_CreateFcn(hObject, eventdata, handles)
   handles.edit1 = hObject;
   guidata(hObject, handles);
   set(hObject,'BackgroundColor','white');


function popupmenu5_Callback(hObject, eventdata, handles)
   val = get(hObject,'Value');
   string_list = get(hObject,'String');
   handles.res = string_list{val};
   guidata(hObject, handles);


function popupmenu4_Callback(hObject, eventdata, handles)
   val = get(hObject,'Value');
   string_list = get(hObject,'String');
```

```
    handles.open = string_list{val};
    guidata(hObject, handles);

function popupmenu3_Callback(hObject, eventdata, handles)
    val = get(hObject,'Value');
    string_list = get(hObject,'String');
    handles.exp = string_list{val};
    guidata(hObject, handles);

function popupmenu2_Callback(hObject, eventdata, handles)
    val = get(hObject,'Value');
    string_list = get(hObject,'String');
    handles.pose = string_list{val};
    guidata(hObject, handles);

function popupmenu1_Callback(hObject, eventdata, handles)
    val = get(hObject,'Value');
    string_list = get(hObject,'String');
    handles.id = string_list{val};
    guidata(hObject, handles);
```

```
classes = {'open', 'sunglasses'}


% the date will be collected into the variables:
X = []; Y = []; ell = 0;


% start in the directory containing only directory names
folders = dir;
% now go into each folder and grab the file names
for i = 3:length(folders), % ignore first two .. and .
   % make sure its a directory
   if folders(i).isdir,
      cd(folders(i).name); % change directory
      files = dir('*_4.pgm');
      for j = 1:length(files),
         % increment out image counter
         ell = ell + 1;
         % read the image
         A = imread(files(j).name);
         files(j).name
         X(:,ell) = A(:);
         % what class is this (part of file name)
         for k=1:length(classes),
                Y(k,ell) = 0.1;
            if ~isempty(findstr(cell2mat(classes(k)),files(j).name)),
                Y(k,ell) = 0.9;
            end
        end
    end
    cd('..'); % go back one folder
  end
end
```

ID_CREATOR.M

```
classes = {'at33', 'ch4f', 'danieln','kawamura' , 'mitchell','saavik','tammo', 'boland', 'cheyer', 'glickman',
'kk49', 'night', 'steffi', 'an2i','bpm','choon','karyadi','megak','phoebe','sz24'}


% the date will be collected into the variables:

X = []; Y = []; ell = 0;


% start in the directory containing only directory names

folders = dir;

% now go into each folder and grab the file names

for i = 3:length(folders), % ignore first two .. and .

  % make sure its a directory

  if folders(i).isdir,

    cd(folders(i).name); % change directory

    files = dir('*_4.pgm');

    for j = 1:length(files),

      % increment out image counter

      ell = ell + 1;

      % read the image

      A = imread(files(j).name);

      files(j).name

      X(:,ell) = A(:);

      % what class is this (part of file name)

      for k=1:length(classes),

          Y(k,ell) = 0.1;

        if ~isempty(findstr(cell2mat(classes(k)),files(j).name)),
                Y(k,ell) = 0.9;
            end
        end
    end
    cd('..'); % go back one folder
  end
end
```

POSE_0_1_CREATOR.M

```
classes = {'right', 'left', 'straight', 'up'};


% the date will be collected into the variables:
X = []; Y = []; ell = 0;
folders = dir;
% now go into each folder and grab the file names
for i = 3:length(folders), % ignore first two .. and .
   % make sure its a directory
   if folders(i).isdir,
      cd(folders(i).name); % change directory
      files = dir('*_4.pgm');
      for j = 1:length(files),
         % increment out image counter
         ell = ell + 1;
         % read the image
         A = imread(files(j).name);
         X(:,ell) = A(:);
         % what class is this (part of file name)
         for k=1:length(classes),
               Y(k,ell) = 0;
            if ~isempty(findstr(cell2mat(classes(k)),files(j).name)),
               Y(k,ell) = 1;
            end
         end
```

```
        end
        cd('..'); % go back one folder
    end
end
```

POSE_CREATOR.M

```
classes = {'right', 'left', 'straight', 'up'};

% the date will be collected into the variables:
X = []; Y = []; ell = 0;
folders = dir;
% now go into each folder and grab the file names
for i = 3:length(folders), % ignore first two .. and .
    % make sure its a directory
    if folders(i).isdir,
        cd(folders(i).name); % change directory
        files = dir('*_4.pgm');
        for j = 1:length(files),
            % increment out image counter
            ell = ell + 1;
            % read the image
            A = imread(files(j).name);
            X(:,ell) = A(:);
            % what class is this (part of file name)
            for k=1:length(classes),
                Y(k,ell) = 0.1;
                if ~isempty(findstr(cell2mat(classes(k)),files(j).name)),
                    Y(k,ell) = 0.9;
                end
            end
        end
        cd('..'); % go back one folder
    end
end
```

FACE_RECOGNIZER.M

```
load id

retrain_number =15;
neuron = 10;
epoch = 100;

X = X / 255; % scale to be within 0 and 1

% partition the data 70/30 cross-validation:
ell = length(Y); I = randperm(ell);
N = ceil(0.7*ell); M = ell - N;
Xv = X(:,I(1:M)); Yv = Y(:,I(1:M));
X = X(:,I(M+1:end)); Y = Y(:,I(M+1:end));
```

```
% initial training just one epoch
[net, error] = vbp(X,Y,[neuron],1,0.3,0.3);

figure

for iteration= 1:retrain_number, iteration
% draw image of weights
  for i = 1:neuron,
    weights = net(2).W(i,2:end);
    weights = weights-min(weights);weights = weights/max(weights)*256;
    subplot(neuron,retrain_number,iteration+(i-1)*retrain_number);
    image(reshape(weights,30,32)),colormap(gray(256))
    set(gca,'yticklabel',[],'xticklabel',[]);
  end

% keep all the error in one data matrix for plotting later
        [net, error] = vbp(X,Y,[neuron],epoch,0.3,0.3,net);
    [net2, error] = vbp(X,Y,[neuron],1,0.3,0.3,net);
        ERROR(:,iteration) = error';
        [net2, error] = vbp(Xv,Yv,[neuron],1,0.3,0.3,net);
        ERRORv(:,iteration) = error';

% check training error
  Yhat = vbp(X, net);
  [dummy, class] = max(Yhat,[],1);
  [dummy, correctclass] = max(Y,[],1);
  training_accuracy(iteration) = (sum(correctclass==class)/length(class)) *100;

% check validation error
  Yhat = vbp(Xv, net);
  [dummy, class] = max(Yhat,[],1);
  [dummy, correctclass] = max(Yv,[],1);
  validation_accuracy(iteration) = sum(correctclass==class)/length(class)*100;
end
```

```
load pose


retrain_number =20;
neuron = 3;
epoch = 100;


X = X / 255; % scale to be within 0 and 1


% partition the data 70/30 cross-validation:
ell = length(Y); I = randperm(ell);
N = ceil(0.7*ell); M = ell - N;
Xv = X(:,I(1:M)); Yv = Y(:,I(1:M));
X = X(:,I(M+1:end)); Y = Y(:,I(M+1:end));


% initial training just one epoch
[net, error] = vbp(X,Y,[neuron],1,0.3,0.3);


figure


for iteration= 1:retrain_number, iteration
% draw image of weights
  for i = 1:neuron,
    weights = net(2).W(i,2:end);
    weights = weights-min(weights);weights = weights/max(weights)*256;
    subplot(neuron,retrain_number,iteration+(i-1)*retrain_number);
    image(reshape(weights,30,32)),colormap(gray(256))
    set(gca,'yticklabel',[],'xticklabel',[]);
  end

% keep all the error in one data matrix for plotting later
        [net, error] = vbp(X,Y,[neuron],epoch,0.3,0.3,net);
    [net2, error] = vbp(X,Y,[neuron],1,0.3,0.3,net);
        ERROR(:,iteration) = error';
        [net2, error] = vbp(Xv,Yv,[neuron],1,0.3,0.3,net);
        ERRORv(:,iteration) = error';

% check training error
  Yhat = vbp(X, net);
  [dummy, class] = max(Yhat,[],1);
  [dummy, correctclass] = max(Y,[],1);
  training_accuracy(iteration) = sum(correctclass==class)/length(class)*100;

% check validation error
  Yhat = vbp(Xv, net);
  [dummy, class] = max(Yhat,[],1);
  [dummy, correctclass] = max(Yv,[],1);
  validation_accuracy(iteration) = sum(correctclass==class)/length(class)*100;
end
```

```
load pose01


retrain_number =10;

neuron = 3;

epoch = 100;


X = X / 255; % scale to be within 0 and 1


% partition the data 70/30 cross-validation:

ell = length(Y); I = randperm(ell);

N = ceil(0.7*ell); M = ell - N;

Xv = X(:,I(1:M)); Yv = Y(:,I(1:M));

X = X(:,I(M+1:end)); Y = Y(:,I(M+1:end));


% initial training just one epoch

[net, error] = vbp(X,Y,[neuron],1,0.3,0.3);


figure


for iteration= 1:retrain_number, iteration

% draw image of weights

  for i = 1:neuron,

    weights = net(2).W(i,2:end);

    weights = weights-min(weights);weights = weights/max(weights)*256;
```

```
   subplot(neuron,retrain_number,iteration+(i-1)*retrain_number);
   image(reshape(weights,30,32)),colormap(gray(256))
   set(gca,'yticklabel',[],'xticklabel',[]);
  end

% keep all the error in one data matrix for plotting later
        [net, error] = vbp(X,Y,[neuron],epoch,0.3,0.3,net);
   [net2, error] = vbp(X,Y,[neuron],1,0.3,0.3,net);
        ERROR(:,iteration) = error';
        [net2, error] = vbp(Xv,Yv,[neuron],1,0.3,0.3,net);
        ERRORv(:,iteration) = error';

% check training error
  Yhat = vbp(X, net);
  [dummy, class] = max(Yhat,[],1);
  [dummy, correctclass] = max(Y,[],1);
  training_accuracy(iteration) = sum(correctclass==class)/length(class)*100;

% check validation error
  Yhat = vbp(Xv, net);
  [dummy, class] = max(Yhat,[],1);
  [dummy, correctclass] = max(Yv,[],1);
  validation_accuracy(iteration) = sum(correctclass==class)/length(class)*100;
end
```

SUNGLASSFACES.M

```
load sunglasses


retrain_number =10;

neuron =2;

epoch = 100;


X = X / 255; % scale to be within 0 and 1


% partition the data 70/30 cross-validation:

ell = length(Y); I = randperm(ell);

N = ceil(0.7*ell); M = ell - N;

Xv = X(:,I(1:M)); Yv = Y(:,I(1:M));

X = X(:,I(M+1:end)); Y = Y(:,I(M+1:end));


% initial training just one epoch

[net, error] = vbp(X,Y,[neuron],1,0.3,0.3);


figure


for iteration= 1:retrain_number, iteration

% draw image of weights

  for i = 1:neuron,

    weights = net(2).W(i,2:end);

    weights = weights-min(weights);weights = weights/max(weights)*256;
    subplot(neuron,retrain_number,iteration+(i-1)*retrain_number);
    image(reshape(weights,30,32)),colormap(gray(256))
    set(gca,'yticklabel',[],'xticklabel',[]);
  end

% keep all the error in one data matrix for plotting later
for i = 1:epoch,
        [net, error] = vbp(X,Y,[neuron],1,0.3,0.3,net);
        ERROR(:,iteration) = error';
        [net2, error] = vbp(Xv,Yv,[neuron],1,0.3,0.3,net);
        ERRORv(:,iteration) = error';
end

% check training error
  Yhat = vbp(X, net);
  [dummy, class] = max(Yhat,[],1);
  [dummy, correctclass] = max(Y,[],1);
  training_accuracy(iteration) = sum(correctclass==class)/length(class)*100;

% check validation error
  Yhat = vbp(Xv, net);
  [dummy, class] = max(Yhat,[],1);
  [dummy, correctclass] = max(Yv,[],1);
  validation_accuracy(iteration) = sum(correctclass==class)/length(class)*100;
end
```

```
% VBP vanilla backpropagation batched training a feed forward multilayer
%     neural network using the log sigmoid tranfer function.
%
% usage: [net,error] = vbp(X, Y, nh, epochs, eta, mu, oldnet);
% where:           X : input layer activation
%                  Y : output layer activation
%                 nh : number of neurons in hidden layers (vector)
%             epochs : number of iterations through all training patterns
%                eta : learning rate \in [0 2]
%                 mu : momentum term \in [0 1]
%
function [net,ep] = vbp(A, T, n, noepochs, eta, mu, oldnet),
% forward sweep for previously trained neural network
  if (nargin == 2),
    for i=1:size(A,2),
      T(1).a = A(:,i);     % the input, then do forward sweep:
      for l=2:length(T(1).n), T(l).a = logsig(T(l).W*[1;T(l-1).a]); end
      net(:,i) = T(end).a; % the neural network output
      ep(i).net = T;
    end
    return;
  end

% number of input/output activations added
```

```
  n = [size(A,1) n size(T,1)];
% number of layers:
  L = length(n);
% number of training samples
  ell = size(A,2);
  if (ell ~= size(T,2)), error('number of X patterns does not match Y'); end
% initialize weight matrices
  net(1).n = n;
  for l=2:L,
    if (nargin == 7), % use last network for initialization
      net(l).W = oldnet(l).W;
      net(l).dWlast = oldnet(l).dWlast; % last weight update for momentum
    else,
      if (l == 2), % special initialization for the first layer
        net(l).W = zeros(n(l),n(l-1)+1);
      else
        net(l).W = 2.0*rand(n(l),n(l-1)+1)-1.0;
      end
      net(l).dWlast = zeros(n(l),n(l-1)+1);
    end
  end
% start training
  for epoch = 1:noepochs,
  % zero the change in weight
    for l = 2:L, net(l).dW = zeros(size(net(l).W)); end
  % loop through each pattern
```

```
for p = 1:ell,
% do a complete forward sweep
  net(1).a = A(:,p); t = T(:,p);
  for l=2:L,
    net(l).netsum = net(l).W*[1;net(l-1).a];
    net(l).a = logsig(net(l).netsum);
  end
% compute root mean square error (for plot)
  e(p) = sqrt((t-net(L).a)'*(t-net(L).a));
% compute signal errors delta (backprop)
  net(L).delta = dlogsig(net(L).netsum).*(t-net(L).a);
  delta = net(L).delta;
  for l = (L-1):-1:2,
    delta = dlogsig([0;net(l).netsum]).*sum((delta*ones(1,n(l)+1)).*net(l+1).W,1)';
    delta = delta(2:end);
    net(l).delta = delta;
  end
% cumulate the weight changes (batched learning)
  for l = 2:L,
    net(l).dW = net(l).dW + net(l).delta * [1;net(l-1).a]';
  end
end
% update the network weights
for l = 2:L,
  net(l).dW  = eta * net(l).dW / ell + mu * net(l).dWlast;
  net(l).W = net(l).W +  net(l).dW;
```

```
      net(l).dWlast = net(l).dW;
    end
    % store mean error per epoch
    ep(epoch) = mean(e);
  end

function [a] = logsig(n),
% log sigmoid transfer function
  a = 1./(1+exp(-n));

function [d] = dlogsig(n),
% log sigmoid transfer derivative function
  a = logsig(n); d = a.*(1-a);
```

ACCURACY.M

```
figure;

plot(1:length(training_accuracy),training_accuracy,1:length(validation_accuracy),validation_accurac
y)

set(get(gca,'XLabel'),'String','number of iterations in 100');

set(get(gca,'YLabel'),'String','Accuracy');

set(get(gca,'Title'),'String','\fontname{times} \it Training(blue) and Validation(green) Accuracy') ;
```

ERROR_PLOT.M

```
figure;

plot(1:length(ERROR),ERROR,1:length(ERRORv),ERRORv)

set(get(gca,'XLabel'),'String','number of iterations in 100');

set(get(gca,'YLabel'),'String','Error');

set(get(gca,'Title'),'String','\fontname{times} \it Training(blue) and Validation(green) Error') ;
```

Project Name	Days	Start	End	27.8	3.9	10.9	17.9	24.9	1.10	8.10	15.10	22.10	29.10	5.11	12.11	19.11	26.11	3.12	10.12
Project Plan 2007	70	27.8	10.4	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

Background research	10	9.7	19.7																
Research	60	27.8	26.10	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦							
Learn how to program MATLAB	60	9.10	8.12							🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	
Basics of Neural Networks	20	10.11	30.11											🟦	🟦	🟦	🟦		
Possible problems	10	1.12	11.12														🟦	🟦	🟦

Design	19	2.10	8.11						🟦	🟦	🟦	🟦	🟦	🟦					
Sunglass	15	2.10	17.10						🟦	🟦	🟦								
Faces	5	17.10	22.10								🟦	🟦							
Pose	7	22.10	29.10								🟦	🟦	🟦						
Viewer	10	29.10	8.11									🟦	🟦	🟦					

Project Name	Days	Start	End	7.1	14.1	21.1	28.1	4.2	11.2	18.2	25.2	3.3	10.3	17.3	24.3	31.3	7.4	14.4	21.4
Research	10	7.1	27.1	🟦	🟦	🟦	🟦												
Learn how to program MATLAB	20	7.1	27.1	🟦	🟦	🟦	🟦												
Coding	10	1.2	7.3					🟦	🟦	🟦	🟦	🟦							
Sunglass	14	1.2	15.2					🟦	🟦										
Faces	5	15.2	20.2							🟦	🟦								
Pose	6	20.2	26.2								🟦	🟦							
Viewer	10	26.2	7.3									🟦	🟦						
Testing and Evaluation	10	7.3	17.3										🟦	🟦					
Accuracy	10	7.3	17.3										🟦	🟦					
Error	10	7.3	17.3										🟦	🟦					
Editing and Documentation	26	15.3	10.4											🟦	🟦	🟦	🟦		

APPENDIX C: USER MANUAL

This tutorial has been developed to get people interest on computer vision and face recognition; in addition, to give you some guidance on how to approach.

The user has to perform the following steps<.

1. open Matlab
2. choose folder containing the Face_Viewer.m file
3. type Face_Viewer in command prompt
4. a window appears
5. choose ID from drop down list
6. select pose from drop down list
7. choose expression from drop down list
8. decides whether per son should wear sunglasses
9. choose resolution from drop down list
10. click “Show image”
11. The desired picture appears.

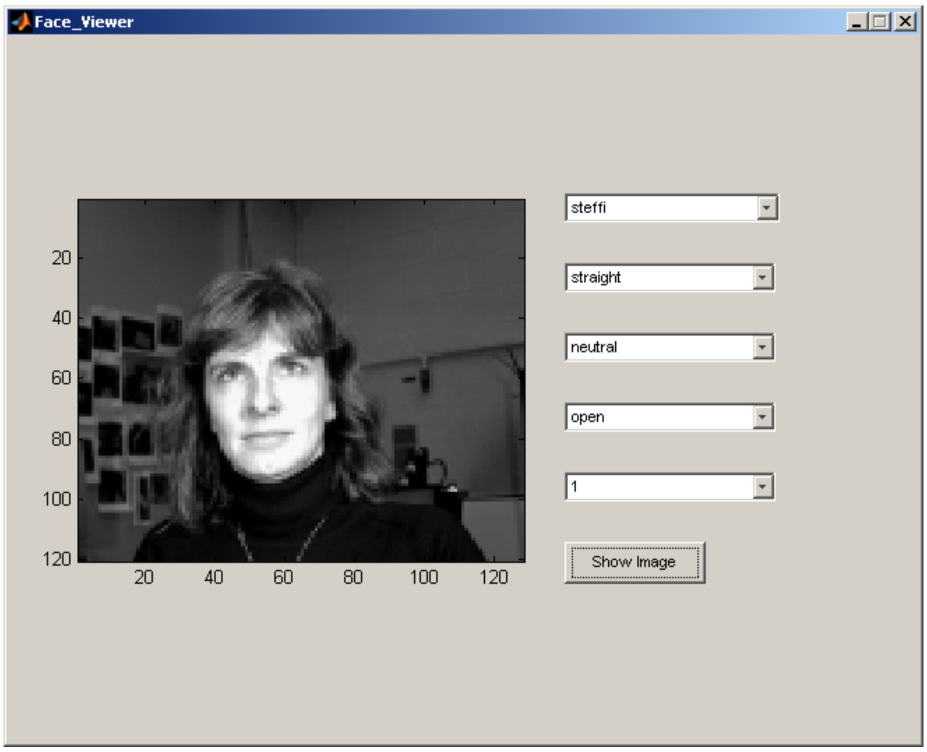


FIGURE 21: FACE VIEWER