UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

COMPUTER SCIENCE

Timo Kleimola

RECOGNIZING LICENSE PLATES FROM DIGITAL IMAGES

Master's thesis for the degree of Master of Science in Technology submitted for inspection in Vaasa, 1st of March 2007.

Supervisor — Professor Jarmo Alander

Instructor — Professor Jarmo Alander

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Timo Kleimola
Diplomityön nimi: Rekisterikilpien tunnistaminen digitaalisista kuvista
Valvojan nimi: Jarmo Alander
Ohjaajan nimi: Jarmo Alander
Tutkinto: Diplomi-insinööri
Laitos: Tietotekniikan laitos
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suunta: Ohjelmistotekniikka
Opintojen aloitusvuosi: 2001
Diplomityön valmistumisvuosi: 2007 **Sivumäärä:** 73

TIIVISTELMÄ:

Tässä diplomityössä tutkitaan mahdollisuutta tunnistaa autojen rekisterikilpiä Nokia N70 -kamerapuhelimen kameralla tuotetusta digitaalisesta kuvasta. Näin tunnistettu rekisterinumero lähetetään GSM-verkon yli Internetissä sijaitsevalle tietokantapalvelimelle SMS-viestinä, josta palautetaan auton tiedot puhelimen näytölle.

Diplomityössä käydään ensin läpi tehtävässä tarvittavien kuvankäsittelyoperaatioiden teoria, josta siirrytään neuroverkon rakenteeseen ja sen opettamiseen backpropagation-algoritmilla. Tämän jälkeen perehdytään siihen, miten tällainen järjestelmä voidaan toteuttaa käytännössä.

Tavoitteena on saavuttaa rekisterinumerojen tunnistamisessa 50%:n luotettavuus, eli keskimäärin joka toisella kerralla rekisterinumero tunnistetaan oikein. Tavoitteena on myös kehittää S60 Platformille tehokkaita, yleiskäyttöisiä kuvankäsittelymetodeja, joita voidaan myöhemmin hyödyntää puhelimien kameroiden sovelluksissa. Tarkoituksena on myös selvittää miten hyvin tällä lähestymistavalla voidaan optista merkkien tunnistusta suorittaa nykyaikaisella mobiilipuhelimella.

AVAINSANAT: Neuroverkot, Symbian OS, signaalinkäsittely, ohjelmointi, optinen merkin tunnistus

UNIVERSITY OF VAASA
Faculty of technology
Author: Timo Kleimola
Topic of the Thesis: Recognizing License Plates from Digital Images
Supervisor: Jarmo Alander
Instructor: Jarmo Alander
Degree: Master of Science in Technology
Department: Department of Computer Science
Degree Programme: Degree Programme in Information Technology
Major of Subject: Software Engineering
Year of the Entering the University: 2001
Year of the Completing the Thesis: 2007 **Pages:** 73

ABSTRACT:

This thesis researches possibility to detect and recognize car license plates from digital images produced with Nokia N70 cameraphone's camera. The recognized plate number is then sent over the GSM-network as an SMS to an Internet database server, from which details of the car are returned into the screen of the phone.

Thesis first introduces reader to image processing operations needed in the process, then moves on to describe the structure of feedforward neural network and teaching it using backpropagation of errors. After this we get into how this system can be implemented in practice.

The aim is to achieve a 50% reliability in recognizing license plates, which means that on average, on every second occasion, the license plate is recognized correctly. The second aim is to develop efficient image processing methods for S60 Platform, that could later be used in applications that utilize a phone camera. Finally, the purpose of the thesis is to find out how well is it possible to do optical character recognition using a modern mobile phone.

KEYWORDS: Neural networks, Symbian OS, signal processing, programming, optical character recognition

TABLE OF CONTENTS

ABBREVIATIONS AND SYMBOLS

API	Application programming interface.
ARM9	An ARM architecture, 32-bit RISC CPU. Used in various mobile phones by many manufacturers.
Epoch	One complete presentation of the entire training set during the learning process of backpropagation algorithm.
Co-operative Multitasking	All application processing occurs in a single thread, and when running multiple tasks, they must co-operate. This co-operation is usually implemented via some kind of a wait loop, which is the case in Symbian OS.
Generalization	A neural network is said to generalize well, when the input-output mapping computed by the network is (nearly) correct for test data never used in the training phase of the network.
Hidden layer	Layer of neurons between output and input layers.
Induced local field	Weighted sum of all synaptic inputs plus bias of a neuron. It constitutes the signal applied to the activation function associated with the corresponding neuron.
Input layer	Consists merely of input signals from the environment. No computation is done in input layer.
Layer	A group of neurons that have a specific function and are processed as a whole. One example is in a feedforward network that has an input layer, an output layer and one or more hidden layers.
LPRS	License plate recognition system.
MMC	The MultiMediaCard is a flash memory card standard. Typically, an MMC card is used in portable devices as storage media.
Neuron	A simple computational unit, that performs a weighted sum on incoming signals, adds a threshold or a bias term to this value resulting in a local induced field and feeds this value to an activation function.
Output layer	Layer of neurons that represents the resulting activation of the whole network when presented with signals from the environment.

RISC	Reduced instruction set computer. A CPU design, in which simplier instructions together with a smaller instruction set is favoured.
Sigmoid function	A strictly increasing, s-shaped function often used as an activation function in a neural network.
Training set	A neural network is trained using a training set. A training set comprises information about the problem at hand as input signals.
Weight	Strength of a synapse between two neurons. Weights may be positive or negative.
Test set	A set of patterns, not included in the training set. Test set, also known as the validation set, is used in between epochs to verify the generalization performance of the neural network.
UI	User interface of an application.
Validation set	See test set.
UML	Unified modeling language. A general purpose modeling language, that includes a standardized graphical notation. UML is used in the field of software engineering for object modeling.

SYMBOLS

$A \cup B$	Union of A and B.
$A \cap B$	Intersection of A and B.
b_k	Bias applied to neuron k.
$d_j(n)$	Desired response for neuron j at iteration n.
$e_j(n)$	The error signal at the output of neuron j at iteration n.
$H(k)$	Histogram of an image, where k is the maximum intensity value of the image.
$y_j(n)$	The output of neuron j at iteration n.
$v_j(n)$	The induced local field of neuron j at iteration n.
$w_{ji}(n)$	Synaptic weight connecting the output of neuron i to input of neuron j at iteration n.
$\delta_j(n)$	Local gradient of neuron j at iteration n.
Δw	Small change applied to weight w.
ε_{av}	Average squared error or sum of squared errors.
$\varepsilon(n)$	Instantaneous value of the sum of squared errors.

η	Learning rate parameter.
σ	Standard deviation.
$\varphi_k(\cdot)$	Nonlinear activation function of neuron k.

1 INTRODUCTION

The number of mobile phones has exploded during the past decade. The rise in ownership has resulted in a constant flood of new ways to utilize the mobile hand set in everyday life. At the time of writing, it was common for a mobile phone to have features such as MP3 playback, e-mail, built-in cameras, camcorders, radio, infrared, GPS navigation and Bluetooth connectivity to name a few.

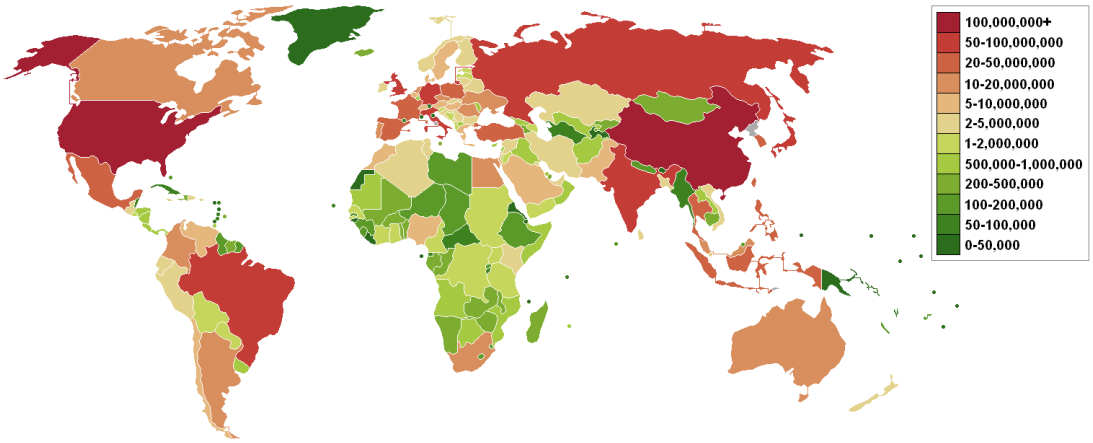

Figure 1. Number of mobile phones across the world (Wik06b).

Currently there are also applications capable of interpreting barcodes from the image taken with the phone's inbuilt camera. People use mobile phones as navigators in their cars or for checking the public transport timetables. This thesis aims to add one application to the long list of applications already available. Recognizing numbers and characters of a license plate and showing some relevant information on a car might not sound like a fascinating application, but it is of academic interest; how well is it currently possible to do optical character recognition using a mobile phone?

I will develop the system for a Nokia N70 mobile phone using the S60 Platform. Recognition engine will consist of an image processing module, a feedforward neural network trained with the backpropagation algorithm and a SMS module for sending license plate information to the AKE's (Ajoneuvorekisterikeskus) SMS service, which will return information on the car in question via SMS.

1.1 General Information on Nokia N70

The new Nokia N70 mobile phone (N70) has a 176×208 pixel TFT-display, which is capable of displaying 262,144 colours. It features two cameras, one in the back of the phone and one on the top. This makes video calls possible between two capable units. The main camera in the back of the phone is a 2 megapixel one, and the one used for video calls is a VGA. The camera on the N70 is a standard CMOS-camera, but the image quality remains pretty decent. Especially the shutter lag has been reduced compared to previous Nokia phones, but it still is sensitive to movements during a shot. Its processor is a 220MHz ARM9E, a 32-bit RISC CPU. It has 22MB of built-in memory with 32MB of RAM.

Its operating system is Symbian OS 8.1a and it uses the S60 Platform 2.8 Feature Pack 3. The S60 Platform is a platform for mobile phones that uses Symbian OS and provides a standards-based development environment for a broad array of smartphones. It is mainly developed by Nokia and licensed by them to other manufacturers like Samsung, Siemens etc. S60 consists of a suite of libraries and standard applications and is amongst the leading smartphone platforms in the world. It was designed from the outset to be a powerful and robust platform for third-party applications. It provides a common screen size, a consistent user interface, a web browser, media player, SMS, MMS and common APIs for JAVA MIDP and C++ programmers. (EB04).

The S60 platform provides a decent set of methods for processing digital images, such as resizing and changing colour depth of an image.

1.2 Introducing Artificial Neural Networks

"An artificial neural network (ANN), also called a simulated neural network (SNN) or commonly just neural network (NN) is an interconnected group of artificial neurons that uses a mathematical or computational model for information processing based on a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network.

In more practical terms, neural networks are non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data." (Wik06a).

The roots of this approach to finding complex relationships between two sets of data lie in the recognition that the human brain computes in an entirely different way from the conventional digital computer. In its most general form, a neural network is designed to model the way in which the brain performs a particular task. This is why the different elements of the network share their names with the ones in the brain.

Neural networks are applicable in almost every situation in which there is a relationship between the input and output variables, even if this relationship is very complex. Applications include among other the following:

- Predicting fluctuations of stock prices and indices based upon various economic indicators.
- Assigning credit. Based on the facts known about loan applicant, such as age, education and occupation, neural network analysis can be used to classify applicant as good or bad credit risk.
- Optical character recognition.

There are many different types of artificial neural networks, such as feedforward, recurrent, stochastic and modular ones, but this thesis introduces only single- and multilayer perceptrons, which are feedforward networks.

1.3 License Plate Recognition

Usually, LPRS' use optical character recognition on images taken with a roadside camera to read the license plates on vehicles. Commonly, they take advantage of infrared lighting to allow the camera to take the image any time of day. A powerful flash is also included in some versions to light up the scene, and to make the offender aware of his mistake. In this application, there will be no (powerful enough) flash nor infrared available, since we will be using a mobile phone to perform this task.

Recognizing license plates using a mobile phone camera is not an easy task to do reliably. There are a number of possible difficulties facing the system in the real world:

- Resolution of the image may be inadequate, or the camera optics may not be of decent quality.

- Poor or uneven lighting, low contrast due to over- or underexposure, shadows and reflections.
- Motion blur in the image due to hand and car movement during the exposure. Shutter speed of the camera should be fast.
- Dirt in the license plates.
- Limited resources of a mobile phone.

Some of these problems can be corrected within the software; for example contrast can be enhanced using histogram equalization and visual noise can be reduced using a median filter. However, there is no escaping dirty license plates or bad optics.

1.4 Related Work

Recognizing license plate's location on an image and recognizing characters using neural networks are very well studied problems. In this section I will shortly describe some approaches that have been taken by researchers to tackle this problem.

There are many applications taking advantage of automated license plate recognition around the world. It is used for example in filling stations to log when a driver drives away without paying. It is also used to control access to car parks (SC99; YAL99), border monitoring, traffic flow monitoring (LMJ90; CCDN+99), law enforcement (DEA90), traffic congestion control (YNU+99) and collecting tolls on pay-per-use roads (Cow95).

(MY06) discusses an algorithm to locate a license plate using multiple Gauss filters followed by morphological image processing. They report an accuracy of 94.33% with 600 test images. (QSXF06) proposes a method in which corners of the license plate characters are used to extract knowledge of the location using an improved Moravec's corner detection algorithm (Mor77). Reported accuracy is 98.42% and the average locating time is 20 ms. Another approach based on morphology is proposed in (SMBR05), and this way of implementation has also been utilized in this thesis. Recently, the work of Matas and Zimmermann (MZ05a; MZ05b) and Porikli and Kocak (PK06) yield impressive results on locating license plates, and generally, text in an image.

Among others, Lecun et al. (LBD+89), Mani et al. (MV96) and Yamada et al. (YKTT89) have utilized neural networks in recognizing characters, including handwritten ones.

2 DIGITAL IMAGE PROCESSING

2.1 Digital Image

Digital image can be presented as a two-dimensional function of the form $f(x, y)$, where x and y are coordinates in the plane. The amplitude of f in each point of the image is called the intensity or gray level. If all these values are finite and discrete, we call the image a digital image. A point, that has a location and an intensity value at that point, is called a pixel. Digital image processing refers to a process of processing digital images using a digital computer.

In this thesis, a digital image is represented using coordinate convention illustrated in figure 2.

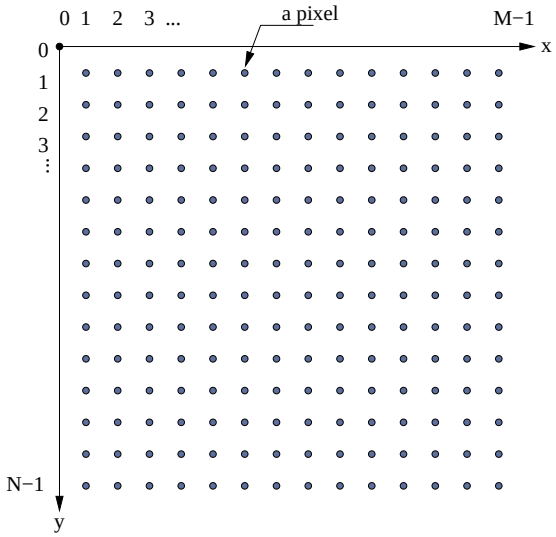

Figure 2. Coordinate convention used in this thesis.

2.2 Neighbourhood

A pixel p at coordinates (x, y) has four horizontal and vertical neighbours at $(x-1, y)$, $(x+1, y)$, $(x, y+1)$ and $(x, y-1)$. This set of pixels is called 4-neighbours of pixel p, denoted $N_4(p)$. If diagonal pixels $(x-1, y+1)$, $(x-1, y-1)$, $(x+1, y+1)$ and $(x+1, y-1)$ are included in this set, the set is called 8-neighbours of p, denoted $N_8(p)$.(GW02).

2.3 Spatial Filtering

Spatial filtering is a process in which a subimage with coefficient values, often referred to as filter, mask, kernel or window (see fig. 3), is slid across a digital image, pixel by pixel. At each point (x, y) the response of the filter is calculated using a predefined relationship.

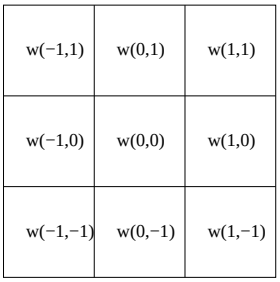

Figure 3. Example of an 3×3 mask with coefficient w.

For linear filtering, the response R, when using 3×3 mask is $R = w(-1,-1)f(x-1, y-1)+w(-1,0)f(x-1,y)+...+w(0,0)f(x,y)+w(1,0)f(x+1,y)+w(1,-1)f(x+1,y-1)$, where $w(s,t)$ is the mask. Note especially, how coefficient $w(0,0)$ coincides with $f(x,y)$ indicating that the mask is centered at (x,y) when the computation of the sum of products is taking place. For a mask of size $m \times n$, we assume that $m = 2a + 1$ and $n = 2b + 1$, where a and b are nonnegative integers. (GW02).

Generally, the response of linear filtering using a mask of size $m \times n$ is given by equation (2.1)

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t), \tag{2.1}$$

where $a = (m-1)/2$ and $b = (n-1)2$. To completely filter an image, this equation has to be applied for $x = 0, 1, 2...M-1$ and $y = 0, 1, 2...N-1$. (GW02).

2.4 Histogram

In context of image processing, histogram refers to a histogram of pixel intensity values in an image. It is a graph representing number of pixels in an image at each intensity level found in that image. In figure 4 there is an 8-bit grayscale image and its pixel distribution

across intensity value range of 0...255. As can be seen from the graph, most of the pixels in this image are in the intensity range of 50..150.

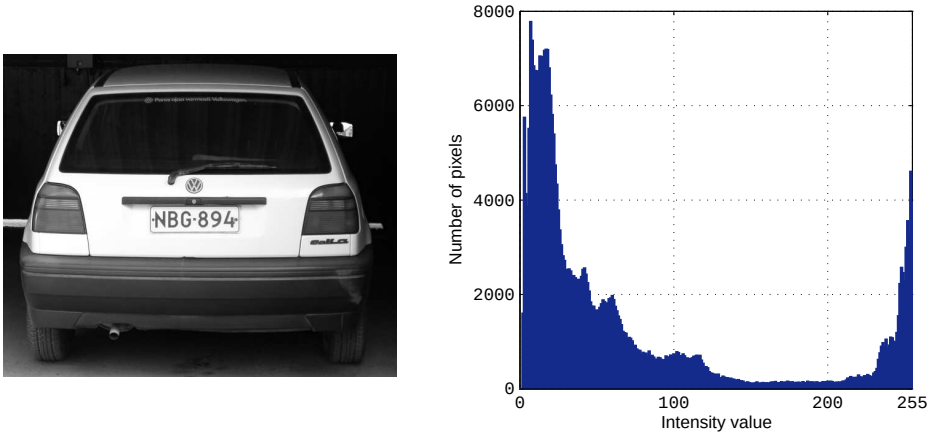

Figure 4. Example of an 8-bit grayscale image of size 1600×1200 pixels and its histogram.

Histograms can be divided in three categories based on how the pixel values in an image are distributed, namely uni-, bi- or multimodal. Unimodal histogram has one peak on its distribution, bimodal has two, multimodal has three or more. Histogram in figure 4 has therefore a multimodal distribution. Unimodal distribution may be the result of a small object in a much larger uniform background, or vice versa. Bimodal histogram is due to an object and background being roughly the same size. Images that have multimodal histograms normally have many objects of varying intensity.

2.4.1 Histogram Equalization

Histogram is a graph representing the distribution of pixel intensity values in an image. Histogram equalization is a process, that redistributes pixels to the largest possible dynamic range. This causes the contrast of the image to improve. Upper limit of this range is dictated by the amount of bits used per colour channel for representing one pixel in an image. This causes the contrast of the image to improve. Equalization is done by mapping every pixel value n of the image into a new value N using equation (2.2).

$$N = (L-1)\frac{\sum_{k=0}^{n} H(k)}{\sum_{k=0}^{L-1} H(k)}, \tag{2.2}$$

where $H(k), k = 0, 1, 2, ..., L-1$ is the histogram of the image and L is the maximum intensity value of the image. (Hut05).

Figure 5 shows how histogram equalization improves the contrast of the image. It also shows how equalization spreads pixel values across the whole intensity range.

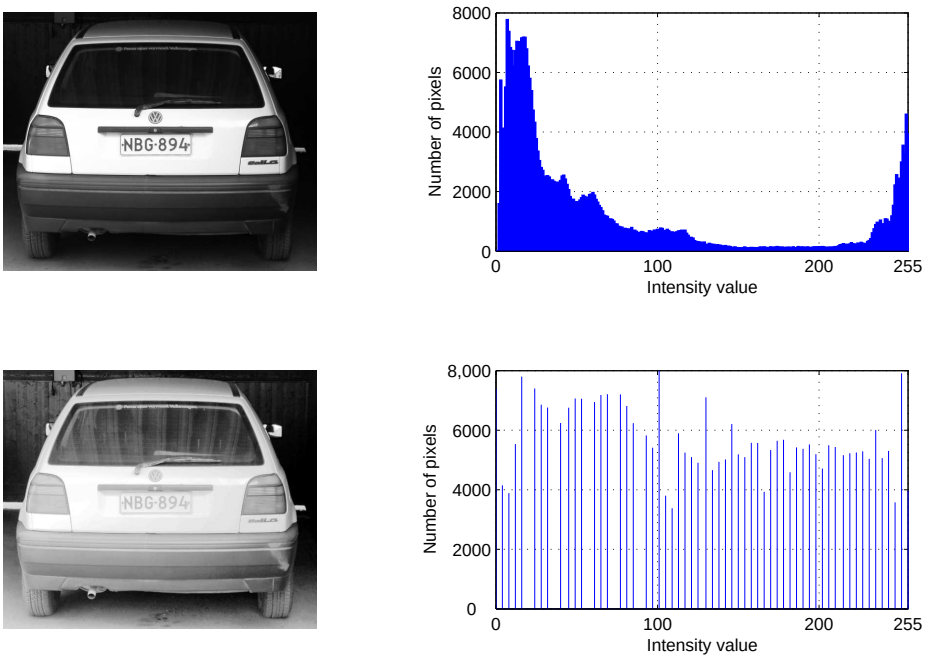

Figure 5. Image and its histogram before and after histogram equalization.

Using standard deviation of the images intensity values, one can programmatically determine whether image would benefit from histogram equalization. Standard deviation describes the spread in data; high contrast image will have a high variance, low contrast image will have a low one. Standard deviation is defined as

$$\sigma = \sqrt{\frac{\sum_{i=0}^{L-1}(x_i - \bar{x})^2}{n}}, \tag{2.3}$$

where $\bar{x}$ is the mean of all intensity values in an image, x_i is ith intensity value, n is the total number of pixels in the image and L is the maximum intensity value possible.

2.5 Edge Detection

Edge detection operators are based on the idea that edge information in an image is found by looking at the relationship a pixel has with its neighbours (Umb05). If the intensity of a pixel is different from its neighbours, it is probable that there is an edge at that point. An edge is therefore a discontinuity in intensity values.

In this thesis, edges are detected using the so called Sobel operator. Sobel operator approximates the gradient by using masks, that approximate the first derivative in each direction. It detects edges in both horizontal and vertical direction and then combines this information to form a single metric. These masks are shown in figure 6.

Vertical edge

−1	−2	−1
0	0	0
1	2	1

Horizontal edge

−1	0	1
−2	0	2
−1	0	1

Figure 6. Sobel filter masks.

Spatial filtering is performed using each of these masks. This produces a set $S = \{r_1, r_2\}$ representing response of the filter at every pixel of the processed image. The magnitude $|\bar{r}|$ of the edge can now be calculated using equation (2.4).

$$\bar{r} = \sqrt{r_1^2 + r_2^2} \tag{2.4}$$

If there's a need to know the direction of the edge, one can use

$$d = \arctan \frac{r_1}{r_2} \tag{2.5}$$

When the application is such, that only horizontal or vertical edges are of interest, one can use the appropriate mask depicted in figure 6 alone. Results of both horizontal and vertical Sobel operators are shown separately in figure 7.

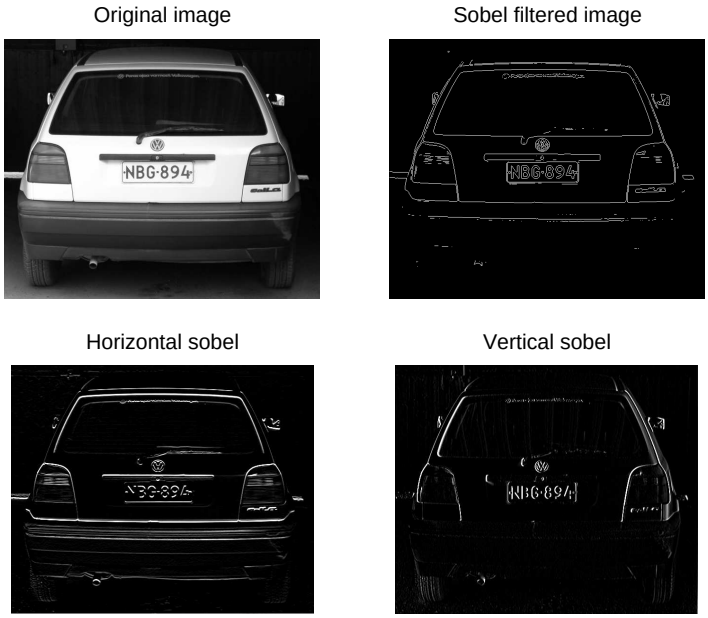

Figure 7. Sobel filtering.

2.6 Thresholding

In its simplest form, thresholding is a process in which each pixel value is compared to some thresholding value T. If the pixel has intensity value that is smaller than or equal to this threshold, its value is changed to either 0 or 1, depending on application. Pixels that are larger than threshold are assigned the other value. The result is a binary image $T(x, y)$:

$$T(x,y) = \begin{cases} 1 & : \quad f(x,y) > T \\ 0 & : \quad f(x,y) \leq T \end{cases}, \tag{2.6}$$

where $f(x, y)$ is the pixel intensity in an image point (x, y). If this operation is carried out as such to the whole image, operation is called global thresholding. If threshold additionally depends on some other property of pixel neighbourhood, for example the average of intensity values in this neighbourhood, thresholding is said to be local. If an image is divided into smaller regions, for each of which separate threshold is calculated, operation is called adaptive thresholding. (GW02; SHB98).

Segmenting using some invented value T normally results in an undesirable output image. This is illustrated in figure 8. A better approach is to first select an initial estimate of the correct threshold value and segment the image using it. This divides the pixels into two classes $C_1 = \{0, 1, 2, ..., t\}$ and $C_2 = \{t, t + 1, t + 2, ..., L - 1\}$, where t is the threshold

Figure 8. Image thresholded using $T = 128$.

value. Normally these classes correspond to the objects of interest and background in an image. The average gray level values μ_1 and μ_2 are then computed for these classes after which new threshold value is calculated using

$$T = \frac{1}{2}(\mu_1 + \mu_2). \tag{2.7}$$

Steps described above are repeated, until the difference in T in successive iterations drops below some predetermined value T_p.

2.7 Morphological Image Processing

Mathematical morphology lends its name from biological concept of morphology, which deals with forms and structures of plants and animals. In morphological image processing, mathematical morphology is used to extract image components, such as boundaries, skeletons etc. The language of mathematical morphology is set theory, which means that it offers unified and powerful approach to many image processing problems. Sets in mathematical morphology represent objects in an image; set of all black pixels constitutes a complete morphological description of the image. In binary images these sets are members of 2D integer space Z^2, where each element consists of coordinates of black pixels in the image. Depending on convention, coordinates of white pixels might be used as well. (GW02; SHB98).

Set theory is not described to any extent in this thesis, since it is assumed, that the reader has a basic knowledge on the subject. Instead, we delve straight into the concepts of dilation and erosion of binary images.

In the following subsections A and B are sets in Z^2 and B denotes a structuring element.

2.7.1 Dilation and Erosion

Dilation and erosion are fundamental operations in morphological processing and as such, provide basis for many of the more complicated morphological operations. Dilation is defined as (GW02; RW96; JH00):

$$A \oplus B = \{z \mid [(\hat{B})_z \cap A] \subseteq A\}, \tag{2.8}$$

In other words, dilation of A by B is the set of all displacements z, such that A and B overlap by at least one element. One of the applications of dilation is bridging gaps, as dilation expands objects. Umbaugh (Umb05) describes dilation in the following way:

- If the origin of the structuring element coincides with a zero in the image, there is no change; move to the next pixel.
- If the origin of the structuring element coincides with a one in the image, perform the OR logical operation on all pixels within the structuring element.

Erosion, a dual of dilation, is defined as (GW02; RW96; JH00):

$$A \ominus B = \{z \mid (B)_z \subseteq A\}. \tag{2.9}$$

Erosion of A by B is therefore the set of points z such, that B, translated by z is contained in A. Erosion is often used when it is desirable to remove minor objects from an image. Size of the removed objects can be adjusted by altering size of the structuring element, B. It can also be used for enlargening holes in an object and eliminating narrow ridges. Umbaugh (Umb05) describes erosion as follows:

- If the origin of the structuring element coincides with a zero in the image, there is no change; move to the next pixel.
- If the origin of the structuring element coincides with a one in the image, any of the one-pixels in the structuring element extend beyond the object in the image, change the one-pixel in the image, whose location corresponds to the origin of the structuring element, to a zero.

These two operations can be combined into more complex sequences. Of these, opening and closing are presented next.

Figure 9. Dilated and eroded image.

2.7.2 Opening and Closing

Like dilation and erosion, opening and closing are important morphological operations. Opening consists of performing dilation after erosion. Closing can be performed by doing erosion after dilation.

Opening is thus

$$A \circ B = (A \ominus B) \oplus B. \tag{2.10}$$

Similarly, closing is defined as

$$A \bullet B = (A \oplus B) \ominus B. \tag{2.11}$$

Generally, opening smoothes the contour of an object, breaks narrow isthmuses and eliminates thin protrusions. Closing on the other hand, also tends to smoothen contours, but unlike opening, fuses narrow breaks and long thin gulfs, fills small holes and fills gaps in the contour (GW02; RW96).

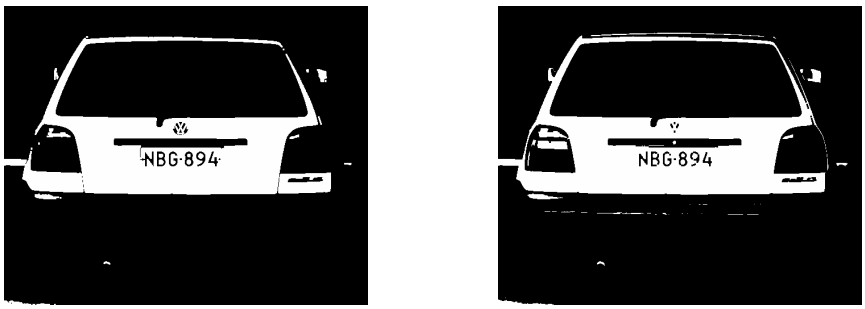

Figure 10. Image after opening and closing.

3 ARTIFICIAL NEURAL NETWORK

”Work on artificial neural networks, commonly referred to as neural networks, has been motivated right from its inception by the recognition that the human brain computes in an entirely different way from the conventional computer.” (Hay99).

Haykin (Hay99) provides a good definition of a neural network:

- ”A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
- 1. Knowledge is required by the network from its environment through a learning process.
- 2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.”

3.1 The Neuron

The simple processing units, that Haykin mentioned are often referred to as neurons. Each neuron consists of a set of synapses, also known as connecting links, an adder and an activation function. Every synapse has its own weight. Adder sums the input signals and weights them by the respective synapses of the neuron. Activation function limits the amplitude of the neuron output. The activation function is often called a squashing function, because it limits the permissible amplitude range of the neuron output signal to some finite range, typically $[0, 1]$ or $[-1, 1]$. Figure 11 depicts a neuron:

Figure 11 also shows an externally applied bias, denoted b_k. For some applications we may wish to increase or decrease the net input of the activation function and this is what bias accomplishes.

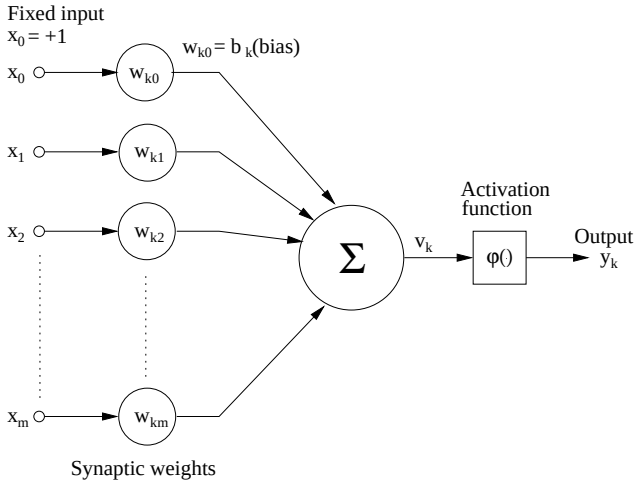

Figure 11. Nonlinear model of a neuron.

Mathematically neuron can be described by equations 3.1 through 3.3. (Hay99).

$$u_k = \sum_{j=0}^{m} w_{kj} x_j \quad \text{and} \tag{3.1}$$

$$v_k = u_k + b_k \tag{3.2}$$

$$y_k = \varphi(u_k + b_k), \tag{3.3}$$

where $x_1, x_2, x_3, ...x_m$ are the input signals, w_{km} are the synaptic weights of neuron k, $\varphi()$ is the activation function, u_k is the adder output, v_k is the induced local field, which is the adder output combined with bias b_k. The result of this is that a graph v_k versus u_k no longer passes through origin; this process is known as affine transformation. As can be seen from figure 11 a bias adds a new input signal with fixed value and synaptic weight that is equal to b_k.

The most popular activation function used in artificial neural network literature seems to be the sigmoid function. Sigmoid function is a strictly increasing function with S-shaped graph. One of the most used sigmoid functions is the logistic function:

$$\varphi(x) = \frac{1}{1 + e^{-x}}. \tag{3.4}$$

Another commonly used sigmoid function is the hyperbolic tangent:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{3.5}$$

Lecun (LBOM98) proposes a modified hyperbolic tangent to be used as an activation

function:

$$f(x) = 1.7159 \tanh(\frac{2}{3}x). \tag{3.6}$$

All these activation functions can be seen in figure 12. Note that the hyperbolic function squashes the neuron output signal to the range of $[-1, 1]$. The logistic function squashes it to the range of $[0, 1]$.

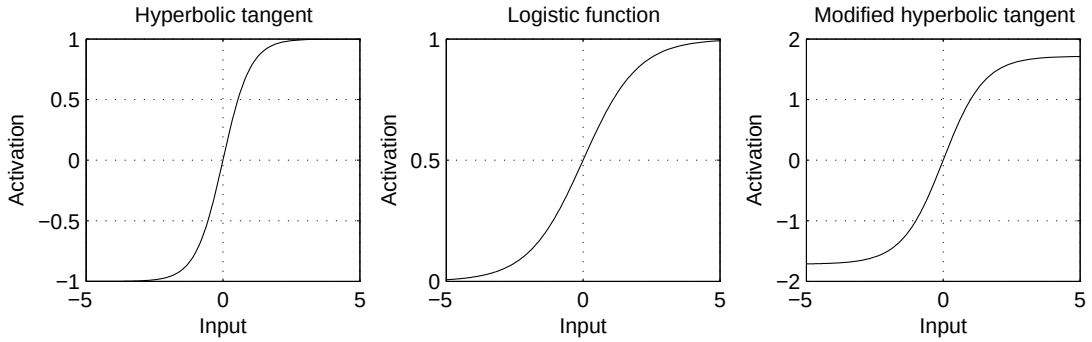

Figure 12. The hyperbolic tangent and the logistic function.

One of the advantages in using the logistic function as an activation function is that its derivative can be obtained rather easily. Derivative of the logistic function (3.4) is:

$$\begin{aligned} \varphi_j'(v_j(n)) &= \frac{e^{v_j(n)}}{(1+e^{v_j(n)})^2} \\ &= \varphi_j(v_j(n))(1-\varphi_j(v_j(n))). \end{aligned} \tag{3.7}$$

Derivative of the hyperbolic tangent is

$$\begin{aligned} \varphi_j'(v_j(n)) &= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\ &= 1 - (\varphi_j(v_j(n)))^2. \end{aligned} \tag{3.8}$$

3.2 Structure of a Feedforward Neural Network

In its simplest form, neural network consists of only input and output layers. This kind of structure is called single-layer perceptron; input layer is really not a layer since its only function is to 'hold' the values inputted into the network. Figure 13 depicts a single-layer perceptron.

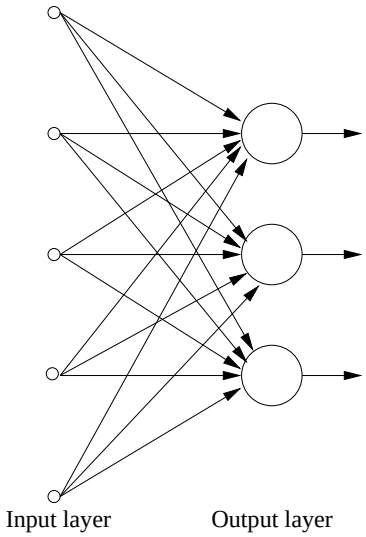

Figure 13. Structure of single-layer perceptron.

Schalkoff (Sch92) describes structure of a feedforward neural network as follows: ”The feedforward network is composed of a hierarchy of processing units, organized in two or more mutually exclusive sets of neurons or layers. The first, or input, layer serves as a holding site for the values applied to the network. The last, or output, layer is the point at which the final state of the network is read. Between these two extremes lies zero or more layers of hidden units. Links, or weights, connect each unit in one layer to those in the next-higher layer. There is an implied directionality in these connections, in that the output of a unit, scaled by the value of a connecting weight, is fed forward to provide a portion of the activation for the units in the next-higher layer.”

Figure 14 shows a two-layer perceptron; it has one hidden layer.

Numerous different neural networks have been invented varying in structure and principle of operation, but this thesis concentrates on multilayer feedforward neural network, which is taught using the backpropagation of errors algorithm.

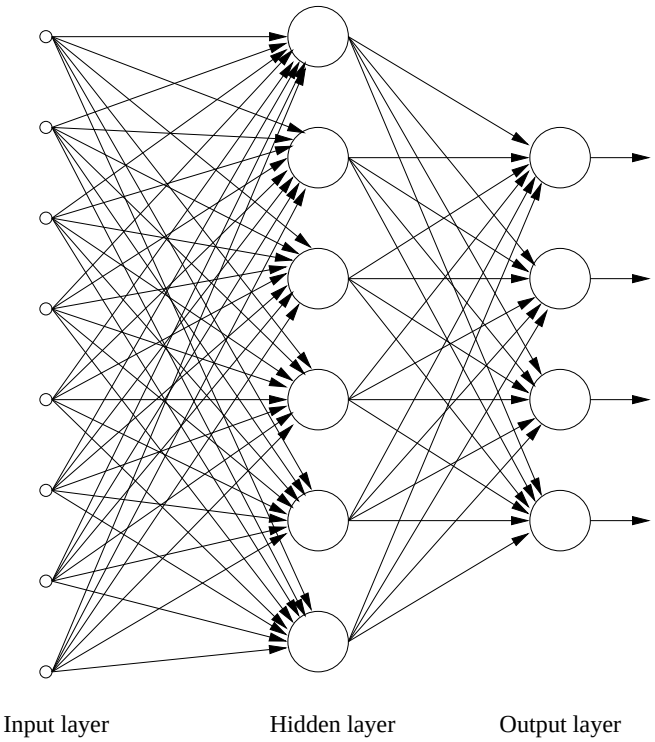

Figure 14. Structure of a two-layer perceptron.

3.3 Backpropagation of Errors

The backpropagation of errors algorithm is a commonly used algorithm for training multilayer perceptrons in a supervised manner. It consists of two passes through the layers of the neural network, referred to as forward pass and backward pass. In the forward pass, an input vector is applied to the neurons of the network, keeping the synaptic weights fixed. In the backward pass, the synaptic weights are adjusted according to an error-correction rule. This means that the actual response of the network is subtracted from the desired response resulting in what is known as the error signal. The error signal is then propagated backward through the network against the direction of synaptic weights; hence the name backpropagation of errors. (Hay99; Sch92).

The error signal $e_j(n)$ at the output of neuron j at iteration n is:

$$e_j(n) = d_j(n) - y_j(n), \tag{3.9}$$

where $d_j(n)$ is the desired response for neuron and $y_j(n)$ is the actual response. When the instantaneous error energy is defined as $\frac{1}{2}e_j^2(n)$, the instantaneous value $\varepsilon(n)$ of the total error energy is obtained by summing the $\frac{1}{2}e_j^2(n)$ over all the neurons in the output layer

of the neural network. (Hay99; Sch92).

$$\varepsilon(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n), \tag{3.10}$$

where C is the set of all the output layer neurons. Here N denotes the total number of elements in the training set. The averaged squared error energy is obtained by summing $\varepsilon(n)$ over all n, and then normalizing the result with respect to the set size N:

$$\varepsilon(n)_{av} = \frac{1}{N}\sum_{n-1}^{N} \varepsilon(n). \tag{3.11}$$

The objective of the learning process is to minimize the averaged squared error energy. In this thesis minimization is done using a simple method of training in which the synaptic weights are updated in accordance with the respective error signals for *each* pattern represented to the network. (Hay99).

The local induced field $v_j(n)$ at the input of activation function associated with neuron j is

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n)y_i(n), \tag{3.12}$$

where m is the total number of inputs applied to neuron j. The output signal of the neuron is therefore

$$y_j(n) = \varphi_j(v_j(n)). \tag{3.13}$$

Illustration of the environment in which the backpropagation algorithm executes can be seen in figure 15.

The backpropagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight w_{ji}, which is proportional to the partial derivative $\partial\varepsilon(n)/\partial w_{ji}(n)$. The correction $\Delta w_{ji}(n)$ applied to synaptic weight $w_{ji}(n)$ is

$$\Delta w_{ji}(n) = -\eta\frac{\partial\varepsilon(n)}{\partial w_{ji}(n)}, \tag{3.14}$$

where η is the learning rate -parameter. This function is referred to as the delta rule. The minus sign in front of the learning rate parameter accounts for the gradient descent in

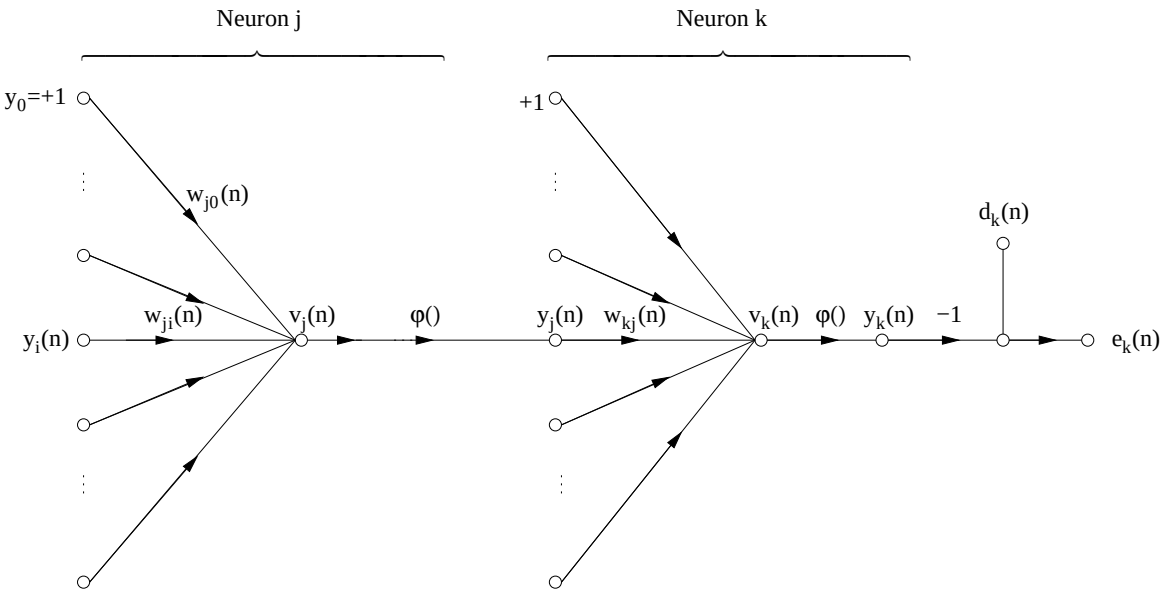

Figure 15. Signal-flow-graph illustrating the details of input neuron j connected to output neuron k.

weight space. This means that the steps taken in the weight space to minimize $\varepsilon(n)$ are taken in the opposite direction of the gradient. (Hay99; Sch92).

The correction can also be written as

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n), \tag{3.15}$$

where $\delta_j(n)$ is the local gradient, denoted as

$$\begin{aligned} \delta_j(n) &= -\frac{\partial \varepsilon(n)}{\partial v_j(n)} \\ &= -\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= e_j(n) \varphi_j'(v_j(n)). \end{aligned} \tag{3.16}$$

The local gradient points in the direction of required changes in synaptic weights. In words, the local gradient $\delta_j(n)$ for output layer neuron j is the product of error signal $e_j(n)$ for that neuron and the derivative $\varphi_j'(v_j(n))$ of the associated activation function. (Hay99).

There is a desired response for each neuron in the output layer, and this makes calculating the gradient easy. For hidden layer neurons there is no desired response, which complicates matters significantly. The local gradient for a hidden layer neuron is shown

in equation 3.17. Detailed derivation can be found in (Hay99).

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n), \tag{3.17}$$

where neuron j is a hidden neuron. What this equation tells us, is that local gradient for a hidden neuron j is a product of derivative of the activation function of that neuron with the sum of products of next layer neurons' local gradients and synaptic weights between neurons j and k.

To summarize what was formulated above, consider equation

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n). \tag{3.18}$$

The correction $\Delta w_{ji}(n)$ is a product of learning rate parameter with local gradient $\delta_j(n)$ and input $y_i(n)$ of neuron j. This equation is called the delta rule. The formulation of the local gradient $\delta_j(n)$ depends on whether associated neuron lies in the hidden or the output layer. If this neuron is an output neuron, $\delta_j(n)$ is the product of the derivative of the activation function $\varphi'(v_j(n))$ and the error signal $e_j(n)$. If the neuron is a hidden neuron, $\delta_j(n)$ is the product of the derivative $\varphi'(v_j(n))$ and the next layer's, or in the case of two-layer perceptron, output layer's δs, weighted with associated synaptic weights between neurons j and k. (Hay99; Sch92).

3.3.1 Error Surface

In order to provide some insight into the matter of how and why different characteristics of the neural network affect the resulting recognition accuracy of the system, we next look at some error surfaces. This is also going to help in understanding different methods that speed up the learning process. In this thesis, an error surface is a three dimensional plot of two network weights as a function of an error signal. Visualizing error surfaces is not an easy task to do in higher than three dimensions, which means that only an error surface of a network with two weights can be accurately visualized. Therefore plots of the error surface in this thesis are 2D-subspaces of the whole surface. This means that error surfaces are a function of a chosen weight pair and the other weights in the network are kept fixed. Also, the characteristics of an error surface depend on the nature of the training data and therefore they differ from problem to problem. (HHS92).

Figure 16 illustrates an error surface of an MLP-network with 5 inputs, 5 hidden layer neurons and one output layer neuron. As stated before, the error surface is a function of two weights, the other being the bias. The network was trained for two epochs on random data between $[-1, 1]$. The result can be seen in figure 16 on the left. The error surface on the right is a result of 10 epochs. One can immediately see, how enlargening the training set affects the error surface. Each element of the training set contributes features to the error surface, since they are combined through the error function.

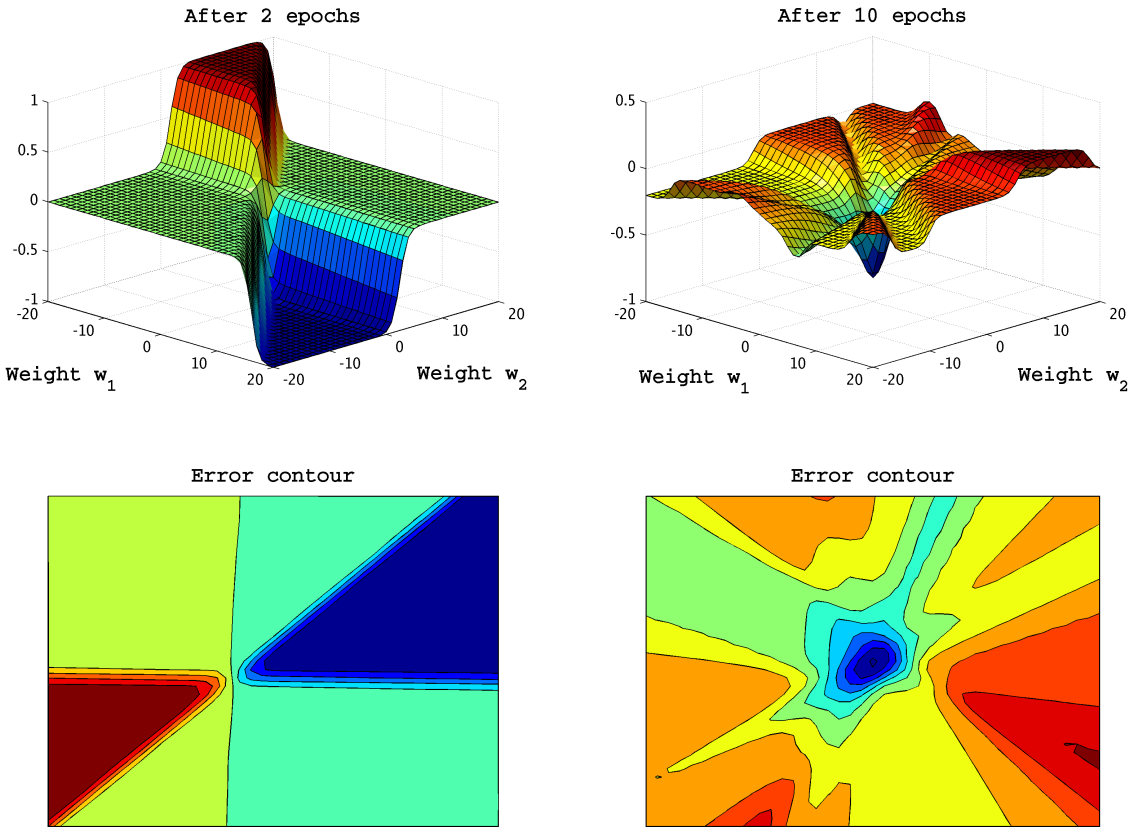

Figure 16. Error surfaces of an MLP with 5 inputs, one output neuron. Network was trained with random data in the range of [-1,1], with backpropagation algorithm for 3 and 10 epochs.

On the error surface on the left, one can see a stair-step appearance with many flat as well as steep regions. In the figure on the left, there are four plateaus, one at the bottom, two in the middle and one at the top. On the bottom plateau, the error energy is at a minimum and all the weight combinations are such that all inputs are classified correctly. The plateaus in the middle correspond to cases where half of the inputs are classified correctly. On the top plateau all of the inputs are misclassified. (HHS92).

3.3.2 Rate of Learning and Momentum

The learning rate parameter η in (3.18) is used to control the size of the change made to the synaptic weights in the network from iteration of the backpropagation algorithm to the next. The smaller the change to η is, the smoother will the trajectory in weight space be. This improvement however, is attained at the cost of slower rate of learning. If we make the η too large in order to speed up the learning process, the resulting changes in synaptic weights could make the network unstable. Finding the optimal rate of learning is a daunting task at best, but luckily, there is a simple method of avoiding danger of instability, but at the same time increasing the rate of learning: modifying the delta rule to include a momentum term

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n), \tag{3.19}$$

where α is usually a positive number called the *momentum constant*. The idea behind applying momentum constant to the delta rule is to make the local gradient to be a weighted sum of successive iterations of the backpropagation algorithm. This filters large oscillations of the gradient and therefore makes the network more stable. (Hay99; Sch92).

3.3.3 Initial Weights

Initializing synaptic weights of the network with good values can be a tremendous help in successful network design and will affect the performance of the network. When weights are initially assigned large values, the network will be driven to saturation quite easily. When this happens, the local gradients in the backpropagation algorithm assume small values and this slows the learning process down. On the other hand, if initial values are small, the algorithm may operate on a very flat area around the origin of the error surface. For these reasons, initializing network weights to either large or small values should be avoided. (Hay99).

How do we choose the initial values then? LeCun (LBOM98) proposes a set of procedures to achieve a better result. First, the training set has to be normalized; convergence is usually faster if the average of the inputs over the whole training set is close to zero. Any shift of the mean input away from zero will bias the update of the weights in a particular direction and therefore slow down the learning process. Convergence can be hastened also by scaling the inputs so that they have approximately the same covariance. The value of the covariance should be matched with that of the sigmoid used.

This procedure should be applied at all layers of the network which means that we also need the outputs of neurons to be close to zero. This can be controlled with the choice of activation function. Sigmoids that are symmetric about the origin (see figure 12) are preferred because they produce outputs that are on average close to zero and therefore make the network converge faster. As stated before, LeCun recommends using the modified hyperbolic tangent (3.6). The constants in this activation function have been selected so that when used with normalized inputs, the variance of the outputs will be close to 1. Now, the initial weights should be randomly drawn from a distribution with mean zero and standard deviation given by $\delta_w = m^{-1/2}$, where m is the number of inputs to the neuron. (LBOM98).

3.4 Teaching the Neural Network

There are two ways for a neural network to learn, namely *supervised* and *unsupervised*. This thesis will only discuss supervised learning, also referred to as *learning with a teacher*. Haykin (Hay99) describes supervised learning as follows: "In conceptual terms, we may think of the teacher as having knowledge of the environment, with that knowledge being represented by a set of *input-output examples*. The environment however is unknown to the neural network of interest. Suppose now that the teacher and the neural network are both exposed to a training vector drawn from the environment. By virtue of built-in knowledge, the teacher is able to provide the neural network with a desired response for that training vector. Indeed, the desired response represents the optimum action to be performed by the neural network. The network parameters are adjusted under the combined influence of the training vector and the error signal. This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network emulate the teacher. When this condition is reached, we may then dispense the teacher and let the neural network deal with the environment by itself."

According to Masters (Mas93), teaching neural network consists of following steps:

- Set random initial values to the synaptic weights of the network.
- Feed training data to the network until the error signal almost stops decreasing.
- Carry out phases 1 and 2 multiple times. The idea behind repetition is to try to find an error surface in which it would be easier to find a better minima.

- Evaluate the performance of the network with a set of previously unseen training examples, called test set. If the difference of the network performance between training set and test set is large, the training set is too small or does not represent the general population. There also might be too much neurons in the hidden layer.

How many hidden layer neurons are needed then? Masters (Mas93) gives some advice on the subject. If there are m neurons in the output layer and n inputs to the network, there has to be $\sqrt{m \times n}$ hidden layer neurons. During the design of the neural network it quickly became obvious, that this method is not going to work for a training set of this size. Eventually the size of the hidden layer became a case in trial and error.

3.4.1 Training Modes

As mentioned earlier, the learning results from the multiple presentations of a training set to the network. One presentation of the complete training set during the learning process is called an *epoch*. This process is maintained until the error signal reaches some predestined minimum value. It is good practice to shuffle the order of presentation of training examples between consecutive epochs. For a given training set, backpropagation learning takes one of the following two forms:

1. The sequential mode, also referred to as *online, pattern,* or *stochastic mode*. In this mode, the network weights are updated after each training example.
2. The batch mode. In the batch mode of operation, weight updating is performed after all the examples in the training set have been presented to the network.

This thesis will concentrate on the former, since sequential mode requires less local storage for each synaptic connection and resources are always scarce in a mobile device. Also, when training data is redundant, which is the case here, sequential mode is able to take advantage of this redundancy because the examples are presented one at a time. Although sequential mode has many disadvantages over batch mode, it is used in this thesis since it is easy to implement and usually provides effective solutions to large and difficult problems. (Hay99).

3.4.2 Criterion for Stopping

There is no well-defined criteria for stopping the operation of backpropagation algorithm as it cannot be shown to converge, but there are some reasonable criteria having some practical merit. This kind of information can be used to terminate the weight adjustments. Perhaps the easiest way is to use some predestined minimum value for the average squared error; when this minimum is reached, the algorithm is terminated. However, this approach may result in premature termination of the learning process. Second approach consists of testing the network's generalization performance after each epoch using a test set as described earlier. When generalization performance reaches adequate level, or when it is apparent that the generalization performance has peaked (see figure 17), the learning process is stopped. This is called the early stopping method. (Hay99; Mit97).

When good generalization performance is kept as a goal, it is very difficult to figure out when to stop the training. If the training is not stopped at the right point, the network may end up *overfitting* the training data. Overfitting deteriorates the generalization performance of the network, and is therefore not desirable. Masters (Mas93) and Mitchell (Mit97) point out, that stopping the learning process early is treating the symptom, not the disease. Both propose a method in which, at first, too few neurons are initialized and trained after which performance of the network is tested with a *test set*, also known as a *validation set*. A validation set consists of inputs unseen by the network during the teaching. If the result is unacceptable, a neuron is added and the whole process is repeated from the start. This is continued until the test set error is at an acceptable level.

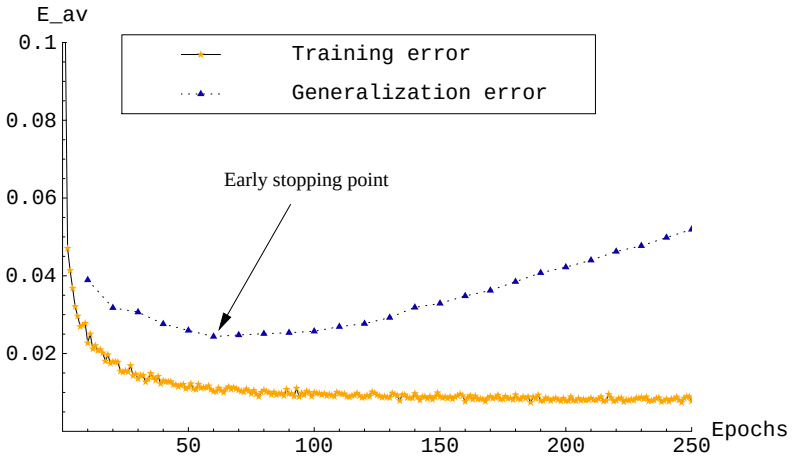

Figure 17. Generalization error against average squared error. The data on the image is simulated and serves only the purpose of illustrating the concept of early stopping point.

Additionally, according to Masters, increasing the size and diversity of the training set will help in learning and this will also be one thing to improve on in the future versions of the recognition system.

4 IMPLEMENTATION USING SYMBIAN C++

This section explains the structure and steps of execution of the license plate recognition software in terms of UML. It also presents some of the image processing algorithms in Symbian C++ and deals with implementing neural network. Before further explaining, to be able to fully grasp the structure of the program, the reader has to adopt some prerequisite information on Symbian OS.

4.1 Prerequisites

First I will shortly introduce the basic and the most commonly used structure for S60 GUI application. After this, Symbian OS DLLs and active objects are shortly explained.

4.1.1 The Structure of an S60 Application

Readers that want more detailed information, can find an exhausting explanation on Series 60 application architecture, and how it derives from and functionally augments Symbian OS framework, in (EB04). In this thesis, presenting the most used anatomy of S60 application will have to suffice.

All S60 UI applications share some functionality. In addition to providing means for a user to interact with the application, they also respond to various system-initiated events (such as redrawing of the screen). The application framework classes that provide this functionality fit into the following high level categories (EB04): Application UI (or AppUi), Application, Document and View.

The AppUi class is a recipient for the numerous framework initiated notifications, such as keypresses and some important system events. The AppUi class will either handle these events itself, or when appropriate, relay the handling to the View class(es) it owns. Classes in this category derive from the framework class CAknAppUi. The Application class serves as a main entry point for the application and delivers application related information back to the framework, like application icons etc. Application class does not deal with the algorithms and data of the application. The Document class is supposed to provide means for persisting application data, and also provides a method for instantiating the AppUi class. The Document class is derived from the framework class CAknDocument.

The View class represent the model's data on the screen. The Model is not a specific class in that it encapsulates the application data and algorithms. This arrangement is known as the *Model-View-Controller* design paradigm (GHJV95; FFSB04).

The basic structure of an S60 application can be seen in figure 18.

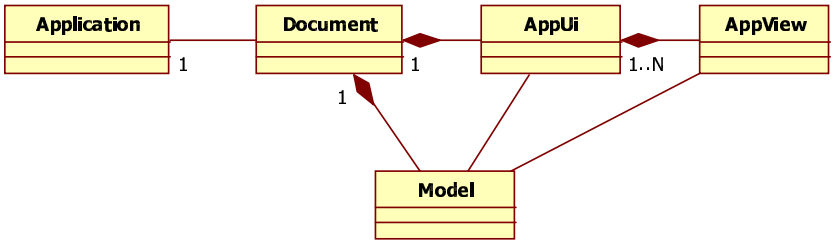

Figure 18. The Basic structure of an S60 application.

4.1.2 Symbian OS DLLs

The executable code of any C++ component in Symbian OS is delivered as a binary package. Packages, that are launched as a new process are referred to as EXEs, and those running in an existing process are called DLLs (dynamically linked libraries) (Sti05).

DLLs consist of a library of compiled C++ code, which can be loaded into a running process in the context of an existing thread (Sti05). When developing application taking advantage of this functionality, rather than linking against the actual library code, the application is linked to library's import table containing dummy functions, which do not contain the details of the implementation. This allows for linking errors to be detected at build time.

There are two main types of DLLs in Symbian OS: shared library DLLs and polymorphic DLLs. One can come across these being called static interface DLLs and polymorphic interface DLLs, which in authors opinion, describe their function better. Of these, only static interface DLLs are discussed further in this thesis.

When executable code that uses a library runs, the Symbian OS loader loads any shared DLLs that it links to as well as any further DLLs needed by those DLLs, doing this until all shared code required by the executable has been loaded. This means that mobile phone's always scarce resources are being saved, because code can be loaded when needed and

unloaded when it becomes useless. Also, it is very desirable that code is being reused through shared libraries, so that they satisfy common functional requirements, that any component in the system may have. (EB04).

Static interface library exports its API methods to a module definition file (.def). This file may list any number of exported methods, each of which is an entry point into the DLL. DLL also releases a header file for other components to compile against and an import library (.lib) to link against, in order to resolve the exported methods.

4.1.3 Active Objects

"Active objects are used on Symbian OS to simplify asynchronous programming and make it easy for you to write code to submit asynchronous requests, manage their completion events and process the result. They are well suited for lightweight event-driven programming, except where a real-time, guaranteed response is required." (Sti05).

Symbian OS is very much an asynchronous operating system. Nearly all system services are provided through servers, which run in their own processes to provide high reliability. APIs to these servers usually provide both synchronous and asynchronous versions of their methods, but when developer wishes to avoid blocking the application's user interface, he/she would normally use the asynchronous one. Therefore most time-consuming operations are made as a request to some asynchronous service provider, such as the file server. The service request method returns immediately, while the request itself is being processed in the background. In the meanwhile, the program continues with other tasks, such as responding to user input and updating the screen. When the asynchronous method has done its task, the program is notified that the request has completed. Such asynchronous systems are prevalent in computing nowadays and there are many ways to implement them, however, only the favoured Symbian OS way is introduced here. (EB04).

"To easily allow Symbian OS program, which typically consists of a single thread within its own process, to issue multiple asynchronous requests and be notified when any of the current requests have completed, a supporting framework has been provided. Cooperative multitasking is achieved through the implementation of two types of object: an *active scheduler* for the wait loop and an *active object* for each task to encapsulate the request and the corresponding handler function." (EB04).

In Symbian OS, it is preferred, that active objects be used. Active objects, which are always derived from class CActive, make it possible for tasks running in the same thread to co-operate, when running simultaneously. This way, there is no need for kernel to change context, when a thread assumes control of the processor after another thread has used its share of processor time, and therefore, this approach preserves phone's resources.

Each concrete class derived from CActive has to define and implement its pure virtual methods DoCancel() and RunL(). DoCancel must be implemented in order to provide the functionality necessary to cancel the outstanding request. Instead of invoking DoCancel() directly, developer should always call the Cancel() method of CActive, which invokes the DoCancel() and also ensures that the necessary flags are set to indicate, that the request is completed. RunL() is the asynchronous event-handler method. This is the method that is called by the Active Scheduler, when the outstanding request has completed. Usually, RunL() consists of a state machine used to control the program's execution steps in desired sequence (EB04).

Active objects are implemented in the following way (EB04):

- Create a class derived from CActive.
- Encapsulate a handle to the service provider as a member data of the class.
- Invoke the constructor of CActive, specifying the task priority.
- Connect to the service provider in your ConstructL() method.
- Call ActiveScheduler::Add() in ConstructL().
- Implement NewL() and NewLC() as normal.
- Implement a method that invokes the request to the asynchronous service, passing iStatus as the TRequestStatus& argument. Call SetActive().
- Implement RunL().
- Implement DoCancel() to handle cancellation of the request.
- Optionally, you can handle RunL() leaves by overriding RunError().
- Implement destructor that calls Cancel() and close all handles on the service providers.

However, when a developer creates a new thread, he/she additionally has to instantiate a CActiveScheduler, install and start it in order to be able to take advantage of this functionality.

4.2 Class Diagram

The software consists of 12 classes. The relationship between them is shown in figure 19. The core classes, explained in 4.1.1, such as the Document and View are omitted for clarity. MLicRecognizerObserver is an abstract super class of CLicRecognizerAppUi, which in turn defines abstract (pure virtual) methods declared in MLicRecognizerObserver. This arrangement is known as the *Observer* design pattern (GHJV95; FFSB04). The observer pattern is usually used to observe the state of an object. In this case, it is used to relay information from the CImageProcLib class to the user interface. Using MLicRecognizerObserver effectively decouples CLicRecognizerAppUi and CImageProcLib from each other thus enforcing the MVC-paradigm.

The CLicRecognizerAppUi is also inherited from another M-class, provided by the ECam library of S60 platform, namely MCameraObserver. MCameraObserver defines methods, that the framework calls, when the phone's camera is being initialized. By implementing these methods in CLicRecognizerAppUi, it is possible to receive information on various states of this initialization process, including readiness of the the camera to take images.

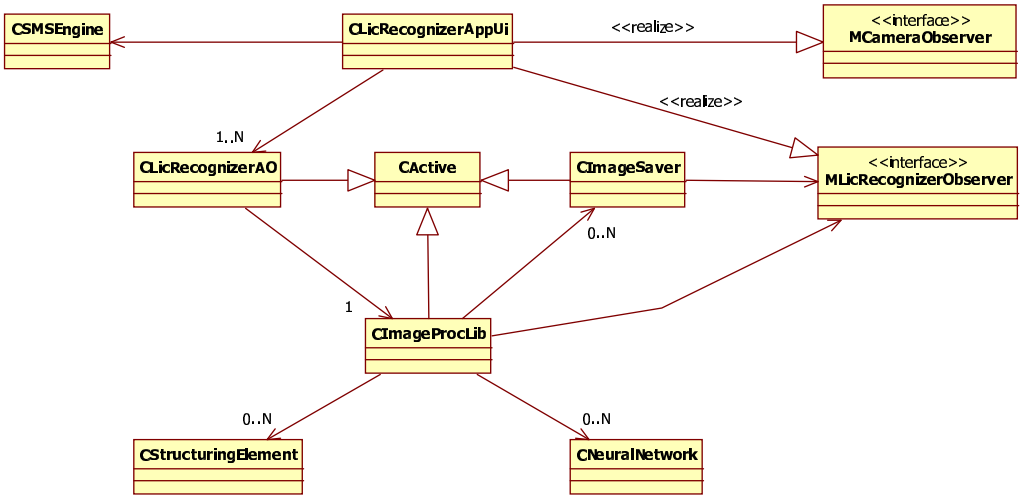

Figure 19. Class diagram for LicRecognizer.

CImageProcLib is a Symbian OS static interface DLL, which provides all the image processing functionality needed in the recognition process. It contains all the image processing methods and spawns a new thread to do the processor intensive calculations. This thread contains an active object, which informs the client class, when the operation has been completed (via TRequestStatus) and also, kills the thread. The image processing algorithms can also be used without the thread synchronously as the DLL exports some of the most useful methods of the class.

CImageSaver is an active object and as such, it is derived from CActive. As its name suggests, its main purpose is to save images to the MMC of the phone. This was mainly used as a helpful debugging (see (KP05)) tool during developing of the image processing algorithms. CLicRecognizerAO is an active object used to start the license plate recognition process and it also provides means to notify the CLicRecognizerAppUi class, that the recognition process has completed.

CNeuralNetwork class is a static interface DLL. It provides an API for training a feedforward neural network using backpropagation of errors. The class also exports methods for executing the network, i.e. for performing *pattern recognition*. In plain English, this class is responsible for interpreting the pixels of the individual characters into plain text. It could be used for any suitable pattern recognition task, not just license plate recognition.

CSMSEngine is a class responsible for sending the SMS, when the license plate has been recognized. CStructuringElement is a class used when performing dilation and erosion.

4.3 Image Processing Algorithms

In order to provide some insight into what is involved in developing S60 image processing algorithms, I will further explain the dilation method I developed for this thesis. This method is capable of binary and grayscale dilation, which is good software design practice; there is no need for separate implentation (code size reduced) and it makes the API easier to use.

```
EXPORT_C TInt CImageProcLib::Dilation(CFbsBitmap* aBitmap,
                                      CStructuringElement* aElement)
        {
        /* Pad image surroundings */
        Pad( aBitmap, aElement, 0 );
```

As described earlier in section 4.1, when a method is to be exported from a DLL, an EXPORT_C statement needs to precede the return value. When declaring the method in the header file, the return value should be preceded with IMPORT_C. The method takes two parameters, a pointer to a CFbsBitmap object and a pointer to a CStructuringElement pointer. Because we are doing dilation, we will pad the surroundings of the image according to the structuring element size. Otherwise the sides of the image will be corrupted.

```
const TInt iWidth = aBitmap->SizeInPixels().iWidth;
const TInt iHeight = aBitmap->SizeInPixels().iHeight;
const TInt c = ( aElement->GetWidth() - 1 )/2;
const TInt r = ( aElement->GetHeight() - 1 )/2;

/* Calculate target image size */
TSize targetSize = TSize(iWidth - (2*c), iHeight - (2*r));

/* Calculate padding bytes for each row */
TInt srcPaddingBytes =
    CFbsBitmap::ScanLineLength( iWidth, EGray256 );
srcPaddingBytes = srcPaddingBytes - iWidth;

TInt dstPaddingBytes =
    CFbsBitmap::ScanLineLength( iWidth + 2 * c, EGray256 );
dstPaddingBytes = dstPaddingBytes - ( iWidth + 2 * c );
```

CFbsBitmap object consists of the so called scanlines, which is an alias for one row of pixels in the image. The memory area representing the scanline in the phone's memory has to be padded so that it is divisible by 4 bytes. Therefore we must calculate the amount of needed padding for both, the source and the destination image, with help from the static ScanLineLength method of the CFbsBitmap class.

```
HBufC8* imageBuffer = HBufC8::NewLC( 2*iWidth*iHeight );
TPtr8 ptr = imageBuffer->Des();

TInt candidate = 0;

aBitmap->LockHeap();
TUint8* start = ( TUint8* ) aBitmap->DataAddress();
```

Here we create an 8-bit heap descriptor and create a pointer descriptor pointing to its data, the resulting dilated image. DataAddress method returns the memory address of the top left corner of the bitmap. In order to ensure, that the heap part of the memory is not reordered by Symbian OS Font and Bitmap server during the process of dilation, we call aBitmap->LockHeap(), which locks the heap.

```
TUint8* tmp;
TInt col = 1;
TInt row = 1;
```

```
const TInt width = aElement->GetWidth();
const TInt height = aElement->GetHeight();

/* Iterate through every pixel in the image */
while( ptr.Length() != ( targetSize.iWidth * targetSize.iHeight ) )
    {
    tmp = start;
    for( TInt i = 0; i < height; i++ )
        {
        for( TInt j = 0; j < width; j++ )
            {
            if ( *start > candidate ) { candidate = *start; }
            start++;
            }
            /* Start of the next row */
            start += ( iWidth - width + srcPaddingBytes );
        }
        /* Append largest value found under SE */
        ptr.Append(candidate);
        candidate = 0;

        /* Return to the upper left corner of the SE */
        start = tmp;

        if ( col == ( targetSize.iWidth ) )
            {
            col = 1;
            start = ( TUint8* ) aBitmap->DataAddress();
            start += row*iWidth + row * srcPaddingBytes;

            for( TInt i = 0; i < dstPaddingBytes; i++ ) { ptr.Append( 0 );}
            row++;
            }
        else { start++; col++; }
    } // While-loop
aBitmap->UnlockHeap();
```

Figure 20 helps the reader to understand, what is going on inside this while-loop. The loop goes through every pixel under the structuring element pixel by pixel, row by row. The largest value found is stored into the heap descriptor named imageBuffer. After this, we return to the upper left corner of the structuring element and increment the memory address by one byte. Once the row has been traversed, we need to skip the padding created using the Pad method in the start of this method and also, add the needed padding to the destination image in order to not to break the 4 byte rule described earlier. The skipped parts are marked by 1 and 2 in the figure.

```
    TInt err = aBitmap->Resize(targetSize);

    if ( err != KErrNone ) { return err; }

    aBitmap->LockHeap();
```

```
start = (TUint8*)aBitmap->DataAddress();

for(TInt i = 0; i < imageBuffer->Size(); i++ )
        {
        *start = ptr[i];
        start++;
        }
aBitmap->UnlockHeap();

CleanupStack::PopAndDestroy( imageBuffer );
return KErrNone;
}
```

Now the imageBuffer descriptor contains the dilated image without the padding. So next we resize the aBitmap instance, to which a pointer was given to as a parameter to the size calculated into variable targetSize earlier. We lock the heap again, get the address of the top left pixel of the aBitmap instance, and fill it with the imageBuffer's data. The only thing left to do is to pop the pointer to the imageBuffer from the cleanup stack and destroy the instance.

Figure 20. Image depicting what needs to be taken into account when dilating an image.

5 RECOGNIZING LICENSE PLATES IN DIGITAL IMAGES

This section begins to uncover the realization of the license plate recognition software. Functionality is explained on a general level beginning with documenting how the license plate can be found from an image and ending in SMS message sent to the AKE. However, let us first see two sequence diagrams, which will show the whole recognition process in compact form.

5.1 Sequence Diagrams

Figures 21 and 22 depict sequence diagrams. Sequence diagrams are a really useful part of UML, which generally shows the interaction between objects over the progression of time. (HM02; Fow04; AN05).

Figure 21 should be self explanatory. It shows the framework instantiating the CLicRecognizerAppUi class, which in turn creates the CLicRecognizerAO instance, which then creates the CImageProcLib instance. As explained in section 4.2, this class starts the recognition process. Figure contains a reference to image processing part of the program execution, which is depicted in 22. One can also see the processing loop, in which each of the detected characters are trimmed, cropped, scaled, stored into a descriptor and fed to the neural network for recognition. After this, a message is shown to the user via user interface containing information on what was recognized. Following this, an SMS message is sent to the AKE.

Figure 22 shows the part of the program flow that does the image processing. First it shows, that it is possible to save all the images the software produces in order to follow the recognition process step by step. This is performed when a truth value called KSaveImages is set to ETrue in the source code. The rest of the figure I'll explain further in the following subsections.

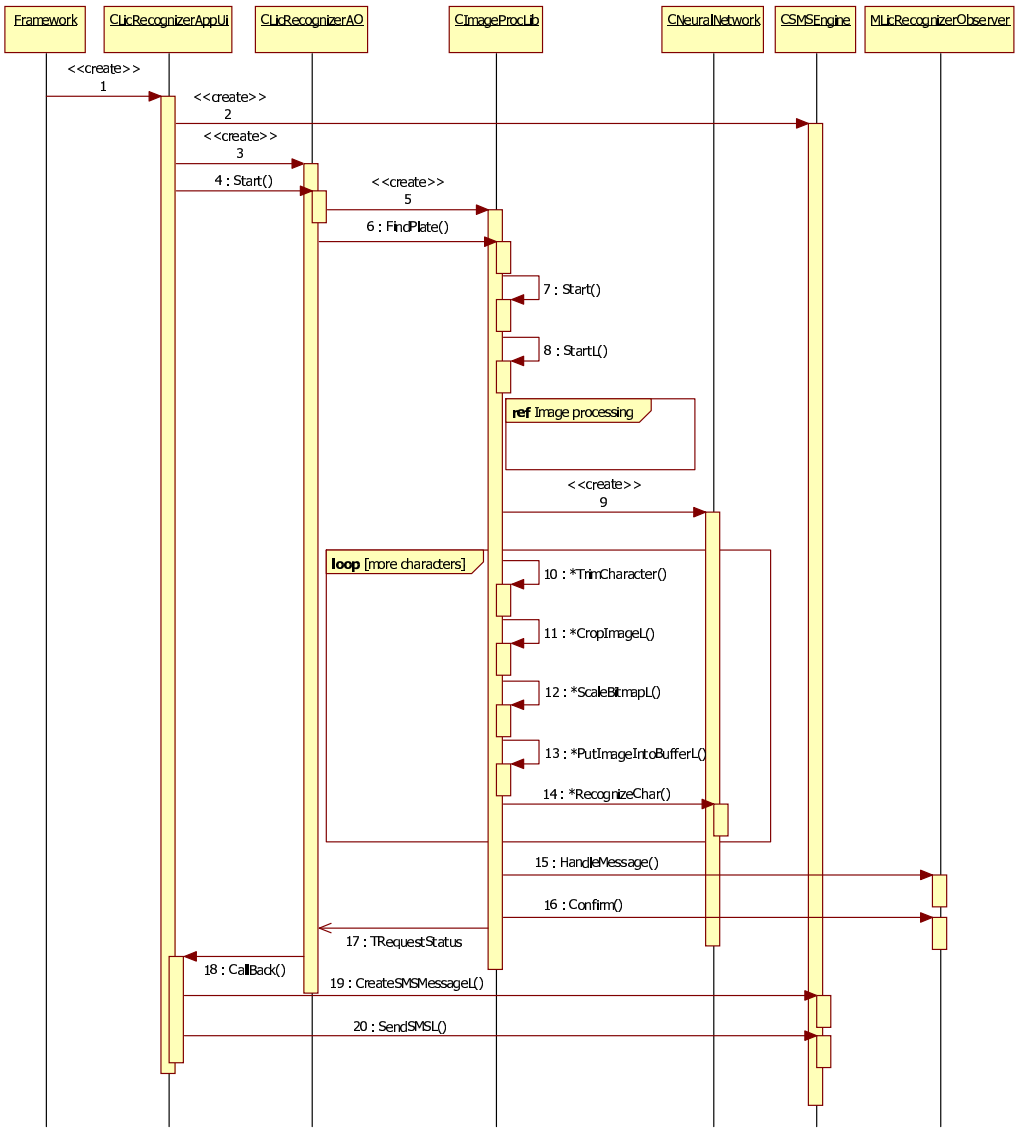

Figure 21. Sequence diagram depicting the whole lifetime of the application.

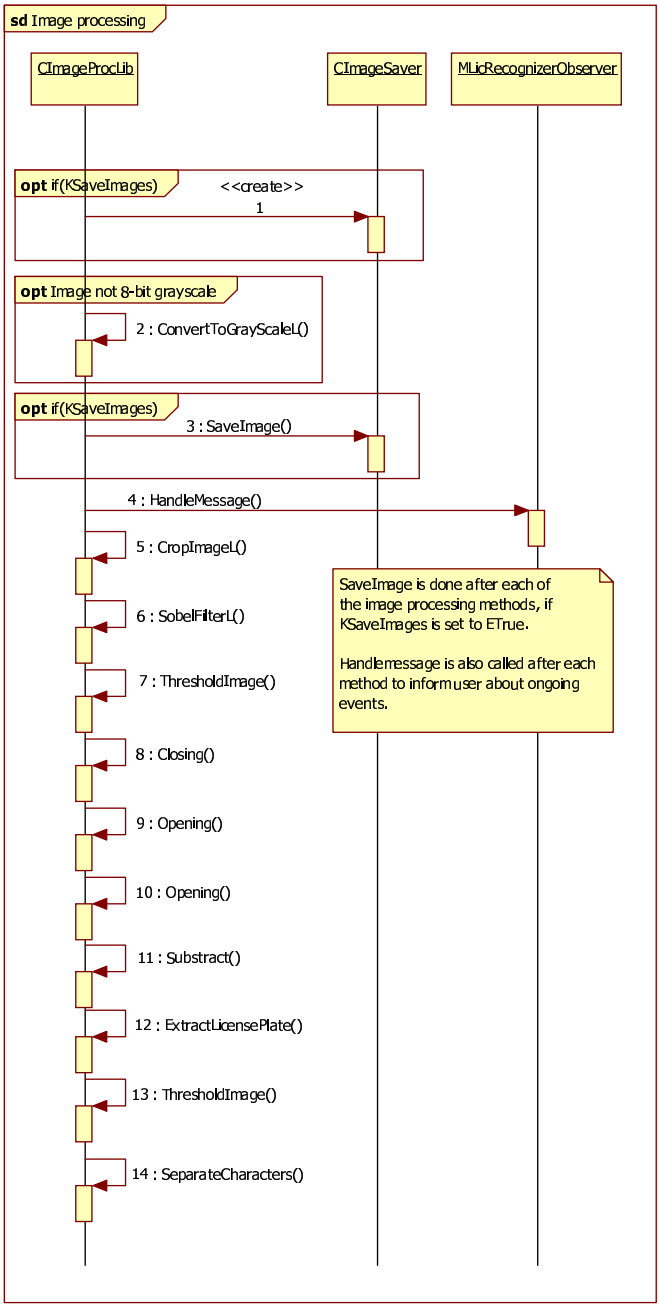

Figure 22. Sequence diagram showing the image processing part of the program flow.

5.2 Locating License Plate in an Image

Firstly, the image captured with the phone's inbuilt camera is converted to 8-bit grayscale, if the colour depth is something else. This makes the processing a whole lot easier and cheaper in terms of processing power. We do not lose any important data converting image to grayscale since we are only interested in white and black areas of the image (license plate). Generally, the system works, when the contrast between the background and characters is good, and the background colour is of lighter tint than the characters. This means that this software is not capable of recognizing license plates of diplomat cars where the background is blue and the characters are white. This limitation derives from the implementation of dilation and erosion in the image processing library of the system.

Due to performance related issues the captured image is then cropped to size of 800x400 pixels. The area which is to be cropped is shown to the user on the screen of the phone, and the user should direct the lens of the camera so, that the license plate is within this area (see figure 23).

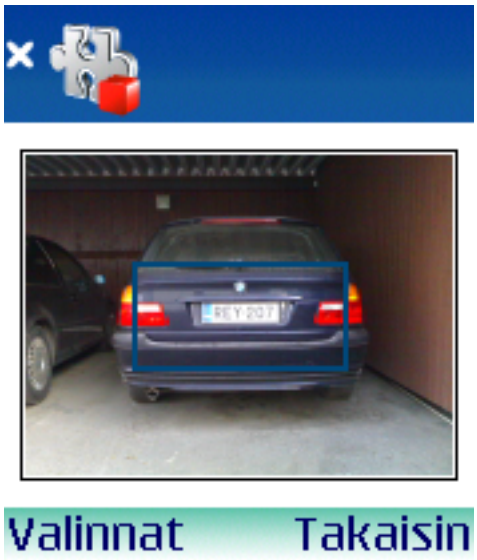

Figure 23. The area into which the user should contain the license plate is shown as blue rectangle.

After cropping, the histogram of the image is equalized (see subsection 2.4.1) to increase the chance of locating the license plate in uneven or otherwise challenging lighting conditions. Next step is to Sobel filter the cropped image using a mask depicted in figure 6. Since we are interested in vertical edges as most of the edges in characters are vertical, we use the vertical edge mask. This yields results shown in figure 24.

Figure 24. Sobel filtered image.

Then the Sobel filtered image is thresholded. The threshold value is calculated following the steps detailed in 2.6.

Figure 25. Image after thresholding.

After thresholding, a closing operation is performed. This is done in order to merge the characters of the license plate together. This prevents the license plate from being cropped during future morphological operations. The size of the structuring element is therefore set to be the maximum space between the characters of the plate. The result is seen in figure 26, in the upper left corner. It is clear, that some processing now needs to be done in order to remove the parts of the image that do not contain the license plate. Therefore, the first opening operation is performed.

I use a structuring element, of which the vertical size is the same as the minimum character height in a license plate. The minimum size has been determined by tests run during application development. The Resulting image can be seen in figure 26, in the upper right corner. One can see, that objects that are smaller than license plate characters have disappeared from the image. There's still some useless blocks in the image, so another opening operation is performed. This time the vertical size of the structuring element is set to be minimum license plate height. As can be seen in figure 26 (lower left image),

this removes all objects, that are smaller than, or of equal size with the license plate. The outcome of the two previous operations are then subtracted from each other, resulting in the image in the lower right corner of the figure 26.

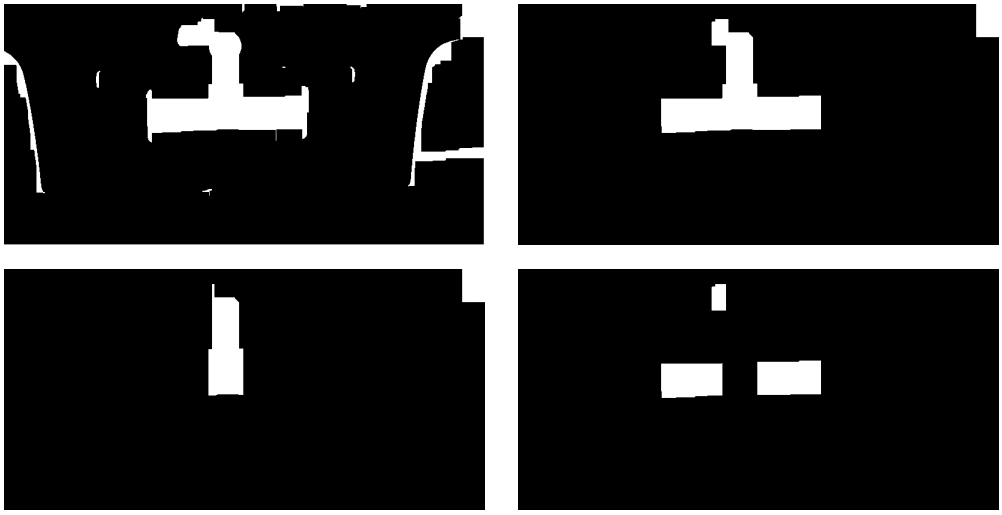

Figure 26. The outcome of morphological operations used in locating the license plate.

The resulting image showing the location of the license plate in the original image can be seen in figure 27. By giving this mask image together with the original grayscale image as a parameter to the ExtractLicensePlate method of the class CImageProcLib, the application crops the license plate area from the original image.

Figure 27. Location of the license plate in the original image.

The resulting image of the license plate is then thresholded, as shown in figure 28.

Figure 28. The result, an extracted plate, thresholded.

5.3 Segmenting Characters

Now that the license plate has been extracted from the captured image, we start cropping individual characters from it. The algorithm that is responsible for locating characters is simple; it calculates standard deviation for each row and column of the extracted license plate. Using this information we can roughly determine where the characters lie in the image.

After this, each character image is trimmed to include only the character using the Trim-Character method of CImageProcLib class. This is followed by scaling the image to the size of 18×28 pixels using the ScaleBitmapL-method.

Figure 29. Separated characters.

After scaling, each image is stored into a descriptor and fed to the neural network for recognition. Some rotation invariance is achieved by storing the image into the descriptor as the number of black pixels in each 2×2 block of the image. This also makes the neural network smaller in size, since there is no need to have an input for each pixel of the image.

5.3.1 Structure of the Network

The neural network consists of input layer, one hidden layer and output layer. The sizes of the layers are 126, 100 and 38 respectively. The size of the input layer is determined by the size of the cropped and scaled individual characters. As mentioned before, the descriptor contains number of black pixels in each 2×2 block of the image, which makes the input layer size $126 = (18 \times 28)/4$.

Size of the hidden layer was determined by trial and error, beginning with too small layer, adding neurons and teaching again. Output layer size is determined by the number of recognition classes an application needs, so in this case, number is 38 since there are 38 numbers and letters in Finnish license plates.

5.3.2 Training the network

Training the network started with a hidden layer of size 25. It quickly became obvious, that a larger one was needed in order for the network to correctly classify license plate characters. Hidden layer size was grown to 25, 50, 75, 100 and finally it was 125. A hidden layer of 100 neurons performs adequately for the training set used. The training set was small, due to resource restrictions brought on by the mobile phone. If we were to train the network to generalize well with many different fonts, the neural network would become too large and the computation of the weights would take too long a time. It would also make training the network cumbersome because of the same reasons. At the same time, this means, that the network is not going to perform as well as it could using a larger and more diverse training set.

The learning rate parameter was calculated for each neuron separately. For hidden layer neurons the learning rate parameter was $1/\sqrt{h}$, where h is the number of inputs to the network. For output layer neurons, the learning rate was calculated using the equation $1/\sqrt{i}$, where i is the number of connections from the hidden layer. This procedure is suggested by LeCun (LBOM98).

Three different momentum values were used in the network, 0.35, 0.5 and 0.65. Figure 30 shows how the momentum affects teaching. It makes the network learn faster and in this case it also yields smaller training error, when trained for 250 epochs.

Due to the smallness of the training set, a network having a 25-neuron hidden layer performed on par with a 100-neuron training set in terms of training error. However, as figure 31 shows, the generalization error has high variance. To recapitulate, this means that the network has learned the training set too accurately. It is unable to perform well on data not previously shown to it. Due to differing lighting conditions, dirt on the plate and image noise, it is highly undesirable for this application to have such a deficiency. This is why more hidden layer neurons were needed although training errors were similar between the two networks.

As mentioned earlier, performance of the network was tested after each epoch using a validation set, which consists of patterns unseen by the network in the training phase. Figure 32 shows how the generalization performance develops as teaching progresses, epoch by epoch, when the hidden layer size is adequate. Picking the right weights for the network's hidden layer was performed by observing which combination of the hidden layer size and momentum yielded best generalization results. In this case, it is a 100-

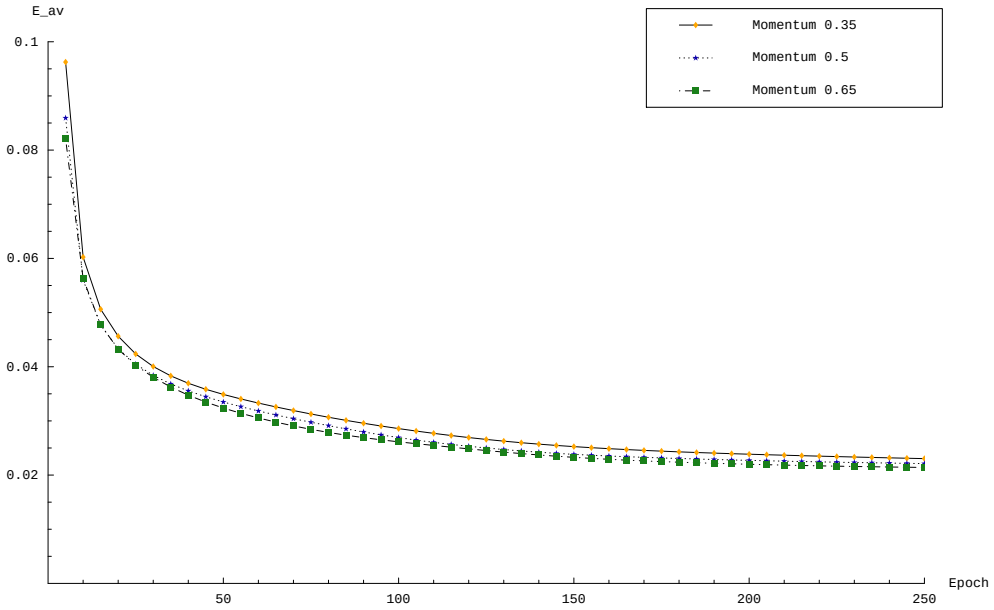

Figure 30. Training errors in a network having hidden layer size of 25 neurons, when momentum range is 0.35, 0.5 and 0.65. The network was trained for 250 epochs.

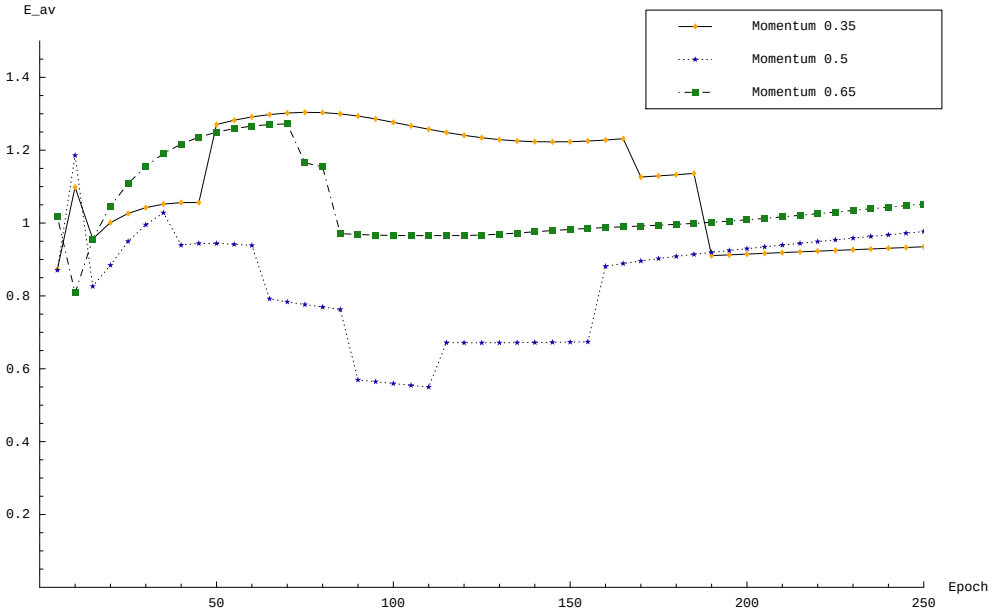

Figure 31. Generalization error for the network having a 25-neuron hidden layer trained for 250 epochs. Error is shown for momentums of 0.35, 0.5 and 0.65.

neuron layer trained for 1000 epochs using a momentum of 0.65.

The network was trained for 1000 epochs because this was enough to make the generalization error stabilize to the level of 0.4.

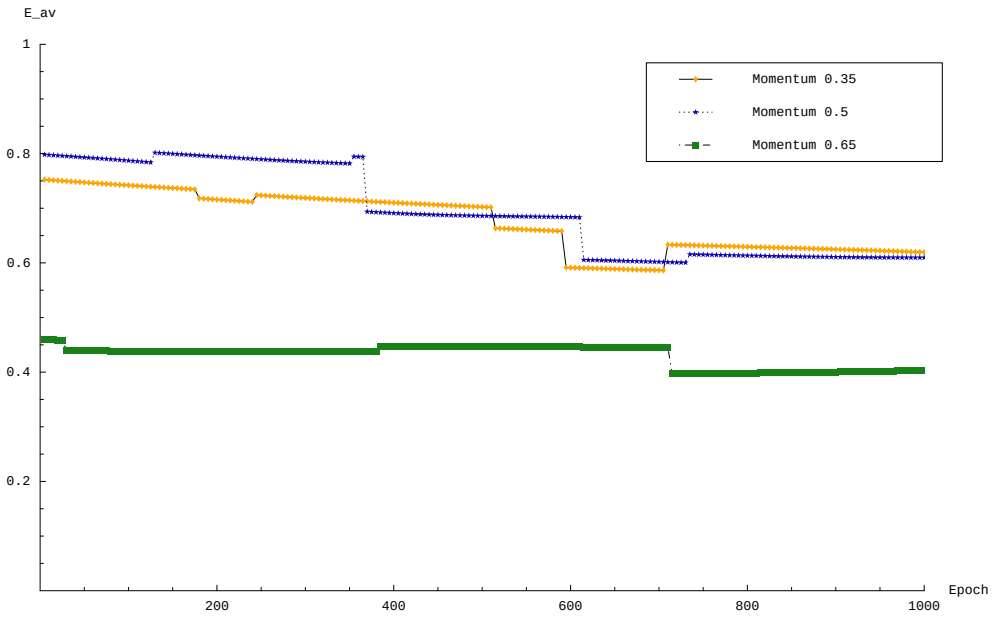

Figure 32. Generalization error for 100-neuron hidden layer, trained for 1000 epochs.

6 RESULTS

The recognition accuracy on the first of the three test images, which is the far-left one in figure 33, can be seen in table 2. The results for the other two images are listed in appendix A. The first column on the left represents the size of the hidden layer used in testing. The second column tells the momentum used in the training phase. The third column shows the recognized characters of the license plate, the fourth column shows the number of epochs taken in the training phase. Finally, the far right column lists the percentage of license plate characters that were recognized correctly.

Figure 33. Test images used in benchmarking the performance of the recognition system.

One can instantly see, that a hidden layer of size 25-75 is not sufficient, since the recognition accuracy is rather poor. In the best case, system recognizes five out of six characters correctly. The average recognition rate for a network with a 25-neuron hidden layer is 59.3%. For a network with a 50-neuron hidden layer, the average is 55.56%. With a hidden layer of 75 neurons, the average is 64.82%.

Accuracy gets better with a larger sized hidden layer, as can be seen in table 3, but starts to deteriorate when size grows beyond 100 neurons.

For this test image, the average recognition rate for a 100-neuron hidden layer is 77.78%. After this, the performance of the system starts to deteriorate as neurons are added to the hidden layer. With a 125-neuron hidden layer, the average drops to 57.41%. As stated earlier, the network with 100 neurons in the hidden layer, trained with a momentum of 0.65 for 1000 epochs was determined to be the optimum for a training set of this size. Thus practice supports the theory.

Table 4 shows the average recognition rates across all the test images. It shows that regardless of the amount of training rounds in the training phase and the momentum used, a 100-neuron hidden layer performs best. The overall recognition accuracy is 67.29%.

Table 2. Recognition accuracy of the system using a test image obtained with the phone's inbuilt camera. This table introduces results for hidden layers of size 25, 50 and 75.

Hidden layer	Momentum	Result	Epochs	Correctly recognized
25	0.35	NBG-888	250	66.67%
25	0.5	LBG-800	250	50.00%
25	0.65	NBG-893	250	83.34%
25	0.35	NBG-888	500	66.67%
25	0.5	LOG-000	500	16.67%
25	0.65	NBG-895	500	83.34%
25	0.35	NBG-888	1000	66.67%
25	0.5	EOG-393	1000	33.34%
25	0.65	?BG-898	1000	66.67%
50	0.35	NBG-895	250	83.34%
50	0.5	BBG-895	250	66.67%
50	0.65	NBG-805	250	66.67%
50	0.35	NBG-224	500	66.67%
50	0.5	NBG-895	500	83.34%
50	0.65	BBG-883	500	50.00%
50	0.35	??G-224	1000	33.34%
50	0.5	VHG-005	1000	16.67%
50	0.65	BBB-893	1000	33.34%
75	0.35	NHG-895	250	66.67%
75	0.5	NBG-895	250	83.34%
75	0.65	NBG-895	250	83.34%
75	0.35	NEE-895	500	50.00%
75	0.5	BBG-898	500	66.67%
75	0.65	NBN-855	500	50.00%
75	0.35	NBG-895	1000	83.34%
75	0.5	BBB-898	1000	50.00%
75	0.65	NBW-855	1000	50.00%

Occasionally, the software fails to locate the license plate in the image correctly. This happens especially, if the image is taken too close or too far away from the car. Also, when there are more than two rectangular objects in the image, the recognition can fail. This is illustrated in figure 34.

Table 3. Recognition accuracy of the system using a test image obtained with the phone's inbuilt camera. This table introduces results for hidden layers of size 100 and 125.

Hidden layer	Momentum	Result	Epochs	Correctly recognized
100	0.35	NBG-895	250	83.34%
100	0.5	NBG-898	250	83.34%
100	0.65	NBG-895	250	83.34%
100	0.35	NBG-855	500	66.67%
100	0.5	NBG-898	500	83.34%
100	0.65	NBG-895	500	83.34%
100	0.35	NBG-555	1000	50.00%
100	0.5	NBG-895	1000	83.34%
100	0.65	NBG-895	1000	83.34%
125	0.35	NBG-898	250	83.34%
125	0.5	NBG-895	250	83.34%
125	0.65	NBG-893	250	83.34%
125	0.35	NBB-895	500	66.67%
125	0.5	NBB-898	500	66.67%
125	0.65	NHG-838	500	50.00%
125	0.35	NEE-855	1000	33.34%
125	0.5	NKK-888	1000	33.34%
125	0.65	EEE-838	1000	16.67%

Table 4. Average recognition accuracy for all three test images, using five different hidden layer sizes.

Hidden layer	Image 1	Image 2	Image 3	Mean
25	59.26%	42.59%	53.71%	51.85%
50	55.56%	62.97%	42.60%	53.71%
75	64.82%	70.37%	55.56%	63.58%
100	77.78%	72.23%	51.85%	67.29%
125	57.41%	74.07%	46.30%	59.26%

Figure 34. On some test images, the software is not capable of locating the license plate. Instead, it locates some other rectangular object in the image as shown here.

7 CONCLUSIONS

To conclude, the questions presented in the beginning of this thesis are answered, together with some suggestions for further development of the license plate recognition software.

7.1 Discussion and Drawn Conclusions

The first objective of this thesis was not reached. The goal was to recognize 50% of license plates correctly. The average character recognition accuracy achieved was 67.29% and only on five occasions was the license plate correctly recognized. This result was achieved on a test images, shown in section 6.

The second goal was to develop a library of useful and efficient image processing algorithms for S60 Platform, that could later be used in similar applications. This goal was met. The library is available as an SIS-installation package along with a header file. There is also a feedforward neural network implementation available shared in a similar manner.

The third objective was to study how well it is possible to do license plate recognition using a modern mobile phone's inbuilt camera in the process. Due to rather large structuring elements in the morphological processing phase, the recognition takes around 20 seconds. This is clearly too much, if the system was to be used in the real world. The image quality of the Nokia N70's camera is not very good. Images have too much noise for this application and the shutter lag is intolerable. It is very easy to take blurry images. This is because the aperture of the camera's lens is nearly nonexistent. Using the approach to the recognition problem taken in this thesis, it is not possible to reliably, and in acceptable time, to recognize license plates in images taken with phone's inbuilt camera.

7.2 Suggestions for Further Development

During the development of the system, many ideas on how to improve the system came afloat.

- The optics of the current camera phones are better. By porting this application to a newer phone, the results would also get better. The increase in processing power in the newer phones would also help.

- The training set should be a lot larger. Unavailability of fonts in a binary matrix form made it a major headache to find teachable material for the neural network.
- The approach to locating a license plate in an image should be changed to a more efficient one. In this application, the structuring elements in the morphological image processing algorithms become large. This increases the number of operations done to the image at each pixel and therefore makes it very slow. One possible approach would be using Moravec's corner detection algorithm for finding license plate and corners of the characters.
- More time should be devoted to training of the neural network. It is a really time-consuming task at best, if one is willing to squeeze the last drop of performance and reliability from the network. It would mean more experimentation with different initial weight values, momentums, layer sizes and epochs. Training data should not be redundant and the training set should be large enough.
- Some of the implemented image processing methods could be reprogrammed in a more efficient manner.
- Skew of the license plate in the image should be removed. Even slightly tilted characters make it more difficult for the neural network to correctly classify them. Teaching the network all possible character orientations would make the network larger, and would therefore require even more effort in the training phase. Also, because the process of locating license plates relies on Sobel filter to find vertical edges in the image, even minor skew is not acceptable.

7.3 Future

The code written for the license plate recognition software could in the future be used for various applications making use of the cameras of the phones. As the optics get better and the phones' processing power increases, more applications become possible.

REFERENCES

[AN05] Jim Arlow and Ila Neustadt. *UML 2 and the Unified Process, Practical Object-Oriented Analysis and Design*. Addison-Wesley, New Jersey, 2. edition, 2005.

[CCDN+99] P. Castello, C. Coelho, E. Del Ninno, E. Ottaviani, and M. Zanini. Traffic monitoring in motorways by real-time number plate recognition. In *Image Analysis and Processing, 1999. Proceedings. International Conference on*, pages 1128–1131, Venice, September 1999.

[Cow95] J.R. Cowell. Syntactic pattern recognizer for vehicle identification numbers. *Image and Vision Computing*, 13:13–19(7), February 1995.

[DEA90] P. Davies, N. Emmott, and N. Ayland. License plate recognition technology for toll violation enforcement. In *Image Analysis for Transport Applications, IEE Colloquium on*, pages 1–7, London, UK, February 1990.

[EB04] Leigh Edwards and Richard Barker. *Developing Series 60 Applications – A Guide for Symbian OS C++ Developers*. Addison-Wesley, Boston, 2004.

[FFSB04] Eric Freeman, Elisabeth Freeman, Kathy Sierra, and Bert Bates. *Head First Design Patterns*. O'Reilly Media, Sebastopol, CA, 1. edition, 2004.

[Fow04] Martin Fowler. *UML Distilled*. Addison-Wesley, Boston, 3. edition, 2004.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, 1. edition, 1995.

[GW02] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, New Jersey, 2. edition, 2002.

[Hay99] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, New Jersey, 2. edition, 1999.

[HHS92] D.R. Hush, B. Horne, and J.M. Salas. Error surfaces for multilayer perceptrons. *IEEE Transactions on Systems, Man and Cybernetics*, 22(5):1152–1161, 1992.

[HM02] Ilkka Haikala and Jukka Märijärvi. *Ohjelmistotuotanto*. Talentum Media, Helsinki, 8. edition, 2002.

[Hut05] Heikki Huttunen. Signaalinkäsittelyn menetelmät, 2005.

[JH00] Bernd Jahne and Horst Haussecker, editors. *Computer Vision and Applications: A Guide for Students and Practitioners*. Academic Press, Inc., Orlando, FL, USA, 2000.

[KP05] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley, Boston, 12. edition, 2005.

[LBD+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989.

[LBOM98] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and K. Muller, editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[LMJ90] R. A. Lotufo, A. D. Morgan, and A. S. Johnson. Automatic number-plate recognition. In *IEE Colloquium on Image Analysis for Transport Applications*, pages 1–6, London, February 1990.

[Mas93] Timothy Masters. *Practical Neural Network Recipes in C++*. Morgan Kaufmann, San Francisco, 1. edition, 1993.

[Mit97] Thomas Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), October 1997.

[Mor77] Hans Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, page 584, August 1977.

[MV96] N. Mani and P. Voumard. An optical character recognition using artificial neural network. In *Systems, Man, and Cybernetics, 1996., IEEE International Conference on*, volume 3, pages 2244–2247, Beijing, October 1996.

[MY06] Zheng Ma and Jingzhong Yang. A license plate locating algorithm based on multiple gauss filters and morphology mathematics. In *SPPRA'06: Proceedings of the 24th IASTED International Conference on Signal Processing, Pattern Recognition, and Applications*, pages 90–94, Anaheim, CA, USA, 2006. ACTA Press.

[MZ05a] J. Matas and K. Zimmermann. Unconstrained licence plate and text localization and recognition. In *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pages 225–230, September 2005.

[MZ05b] Jiri Matas and Karel Zimmermann. A new class of learnable detectors for categorisation. In Heikki Kälviäinen, Jussi Parkkinen, and Arto Kaarna, editors, *SCIA '05: Proceedings of the 14th Scandinavian Conference on Image Analysis*, volume 3540 of *LNCS*, pages 541–550, Heidelberg, Germany, June 2005. Springer-Verlag.

[PK06] F. Porikli and T. Kocak. Robust license plate detection using covariance descriptor in a neural network framework. In *Video and Signal Based Surveillance, 2006. AVSS '06. IEEE International Conference on*, pages 107–107, Sydney, Australia, November 2006.

[QSXF06] Zhong Qin, Shengli Shi, Jianmin Xu, and Hui Fu. Method of license plate location based on corner feature. In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, volume 2, pages 8645–8649, June 2006.

[RW96] Gerhard X. Ritter and Joseph N. Wilson. *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[SC99] T. Sirithinaphong and K. Chamnongthai. The recognition of car license plate for automatic parking system. In *Signal Processing and Its Applications, 1999. ISSPA '99. Proceedings of the Fifth International Symposium on*, volume 1, pages 455–457, Brisbane, Qld., August 1999.

[Sch92] Robert J. Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches*. John Wiley and Sons, New York, 1. edition, 1992.

[SHB98] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis and Machine Vision*. PWS Publishing, Pacific Grove, CA, 2. edition, 1998.

[SMBR05] P. V. Suryanarayana, S. K. Mitra, A. Banerjee, and A. K. Roy. A morphology based approach for car license plate extraction. In *INDICON, 2005 Annual IEEE*, pages 24–27, December 2005.

[Sti05] Jo Stichbury. *Symbian OS Explained*. John Wiley and Sons, 1. edition, 2005.

[Umb05] Scott E. Umbaugh. *Computer Imaging – Digital Image Analysis and Processing*. CRC Press, Boca Raton, Florida, 2005.

[Wik06a] Wikipedia. Artificial neural network — Wikipedia, the free encyclopedia, 2006. [Online; accessed 18-July-2006].

[Wik06b] Wikipedia. Mobile phone — Wikipedia, the free encyclopedia, 2006. [Online; accessed 18-July-2006].

[YAL99] N. H. C. Yung, K. H. Au, and A. H. S. Lai. Recognition of vehicle registration mark on moving vehicles in an outdoor environment. In *Intelligent Transportation Systems, 1999. Proceedings. 1999 IEEE/IEEJ/JSAI International Conference on*, pages 418–422, Tokyo, October 1999.

[YKTT89] K. Yamada, H. Kami, J. Tsukumo, and T. Temma. Handwritten numeral recognition by multilayered neural network with improved learning algorithm. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 259–266, Washington, DC, June 1989.

[YNU+99] K. Yamaguchi, Y. Nagaya, K. Ueda, H. Nemoto, and M. Nakagawa. A method for identifying specific vehicles using template matching. In *Intelligent Transportation Systems, 1999. Proceedings. 1999 IEEE/IEEJ/JSAI International Conference on*, pages 8–13, Tokyo, October 1999.

A Appendices

A.1 Test Results

Table 5. Recognition accuracy of the system using a test image obtained with the phone's inbuilt camera. This table introduces results for hidden layers of size 25-125 on the second test image.

Hidden layer	Momentum	Result	Epochs	Correctly recognized
25	0.35	NHG-884	250	66.67%
25	0.5	QEG-000	250	16.67%
25	0.65	NNG-857	250	50.00%
25	0.35	NHG-888	500	50.00%
25	0.5	EOG-000	500	16.67%
25	0.65	UUG-858	500	33.34%
25	0.3	NSG-884	1000	66.67%
25	0.5	EEG-338	1000	16.67%
25	0.65	NBG-878	1000	66.67%
50	0.35	NPG-894	250	83.34%
50	0.5	NHG-894	250	83.34%
50	0.65	NBG-894	250	100.00%
50	0.35	GGG-444	500	33.34%
50	0.5	HHG-808	500	33.34%
50	0.65	NBG-884	500	83.34%
50	0.35	?HG-224	1000	33.34%
50	0.5	VHG-000	1000	16.67%
50	0.65	NBG-894	1000	100.00%
75	0.35	NHG-894	250	83.34%
75	0.5	NBG-894	250	100.00%
75	0.65	NBG-895	250	83.34%
75	0.35	NHG-894	500	83.34%
75	0.5	NBG-894	500	100.00%
75	0.65	NNG-855	500	50.00%
75	0.35	NHG-890	1000	50.00%
75	0.5	BBG-894	1000	83.34%
75	0.65	QQQ-000	1000	0.00%
100	0.35	NGG-894	250	83.34%

100	0.5	NHG-894	250	83.34%
100	0.65	NEG-894	250	83.34%
100	0.35	NGG-855	500	50.00%
100	0.5	NHG-894	500	83.34%
100	0.65	NEG-894	500	83.34%
100	0.35	NQG-555	1000	33.34%
100	0.5	NGG-894	1000	83.34%
100	0.65	?EG-894	1000	66.67%
125	0.35	NBG-898	250	83.34%
125	0.5	NBG-894	250	100.00%
125	0.65	NHG-894	250	83.34%
125	0.35	NHG-898	500	66.67%
125	0.5	NBG-898	500	83.34%
125	0.65	NHG-884	500	66.67%
125	0.35	NEG-894	1000	83.34%
125	0.5	NKG-888	1000	50.00%
125	0.65	EEG-884	1000	50.00%

Table 6. Recognition accuracy of the system using a test image obtained with the phone's inbuilt camera. This table introduces results for hidden layers of size 25-125 on the third test image.

Hidden layer	Momentum	Result	Epochs	Correctly recognized
25	0.35	ENV-535	250	66.67%
25	0.5	EOV-030	250	33.34%
25	0.65	EMV-535	250	83.34%
25	0.35	EOV-535	500	66.67%
25	0.5	EOO-030	500	16.67%
25	0.65	EUV-535	500	66.67%
25	0.35	HNV-035	1000	50.00%
25	0.5	EEW-333	1000	16.67%
25	0.65	EMV-535	1000	83.34%
50	0.35	ELV-535	250	66.67%
50	0.5	BHV-535	250	83.34%
50	0.65	HBV-335	250	50.00%
50	0.35	PGV-535	500	66.67%
50	0.5	HHV-535	500	66.67%
50	0.65	EEB-333	500	16.67%
50	0.35	H??-424	1000	0.00%
50	0.5	HLV-000	1000	16.67%
50	0.65	EBB-333	1000	16.67%
75	0.35	ENV-535	250	66.67%
75	0.5	EEB-535	250	50.00%
75	0.65	SNV-535	250	66.67%
75	0.35	ENV-535	500	66.67%
75	0.5	EBB-535	500	50.00%
75	0.65	NNN-535	500	50.00%
75	0.35	ENV-535	1000	66.67%
75	0.5	EBB-535	1000	50.00%
75	0.65	SQQ-505	1000	33.34%
100	0.35	ENV-535	250	66.67%
100	0.5	ENB-535	250	50.00%
100	0.65	EBB-535	250	50.00%
100	0.35	ENV-535	500	66.67%
100	0.5	ENB-535	500	50.00%

100	0.65	EBB-535	500	50.00%
100	0.35	EGV-555	1000	50.00%
100	0.5	ENB-535	1000	50.00%
100	0.65	EEB-585	1000	33.34%
125	0.35	ENB-535	250	50.00%
125	0.5	BNV-535	250	83.34%
125	0.65	ENV-338	250	33.34%
125	0.35	ENB-535	500	50.00%
125	0.5	NNV-535	500	66.67%
125	0.65	EHH-838	500	16.67%
125	0.35	ENE-535	1000	50.00%
125	0.5	NNK-535	1000	50.00%
125	0.65	EEE-838	1000	16.67%

A.2 The Training Set

0 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 3, 2, 3, 4, 4, 0, 1, 4, 3, 0, 0, 0, 3, 4, 1, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 1, 4, 3, 0, 0, 0, 3, 4, 1, 0, 4, 4, 3, 2, 3, 4, 4, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

1 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

2 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 4, 3, 1, 0, 0, 0, 1, 4, 4, 2, 4, 4, 1, 0, 0, 4, 3, 0, 0, 0, 4, 3, 0, 0, 4, 2, 0, 0, 0, 2, 4, 0, 0, 3, 1, 0, 0, 0, 4, 3, 0, 0, 0, 0, 0, 1, 4, 4, 1, 0, 0, 0, 0, 1, 4, 4, 1, 0, 0, 0, 0, 1, 4, 3, 1, 0, 0, 0, 0, 0, 3, 4, 1, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 3, 2, 2, 2, 2, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0

3 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 4, 4, 4, 4, 4, 4, 3, 0, 0, 2, 2, 2, 3, 4, 4, 1, 0, 0, 0, 0, 0, 2, 4, 3, 0, 0, 0, 0, 0, 1, 4, 4, 1, 0, 0, 0, 0, 0, 4, 4, 4, 2, 0, 0, 0, 0, 3, 4, 4, 4, 4, 3, 0, 0, 0, 0, 0, 0, 1, 4, 4, 1, 0, 0, 0, 0, 0, 0, 2, 4, 2, 1, 4, 1, 0, 0, 0, 2, 4, 2, 1, 4, 4, 0, 0, 0, 2, 4, 2, 0, 3, 4, 4, 2, 3, 4, 4, 0, 0, 0, 3, 4, 4, 4, 3, 1, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0

4 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 1, 4, 1, 0, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0, 1, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 0, 0, 0, 0, 0, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 0, 0, 2, 1, 0, 0, 0, 4, 3, 0, 0, 4, 2, 0, 0, 1, 4, 2, 0, 0, 4, 2, 0, 0, 3, 4, 4, 4, 4, 4, 4, 2, 0, 2, 2, 2, 2, 2, 4, 3, 1, 0, 0, 0, 0, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

5 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 4, 4, 4, 4, 2, 0, 0, 0, 3, 4, 2, 2, 2, 1, 0, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 4, 4, 3, 1, 0, 0, 0, 4, 4, 2, 2, 4, 4, 2, 0, 0, 0, 0, 0, 0, 0, 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 2, 1, 2, 0, 0, 0, 0, 0, 0, 4, 2, 1, 4, 3, 0, 0, 0, 3, 4, 0, 0, 3, 4, 4, 2, 3, 4, 2, 0, 0, 0, 2, 4, 4, 4, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

6 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 4, 1, 0, 0, 0, 0, 3, 4, 4, 2, 1, 0, 0, 0, 3, 4, 3, 0, 0, 0, 0, 0, 1, 4, 3, 0, 0, 0, 0, 0, 0, 3, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 3, 0, 0, 2, 4, 4, 4, 4, 4, 4, 3, 0, 2, 4, 3, 0, 0, 0, 4, 4, 0, 1, 4, 2, 0, 0, 0, 2, 4, 0, 0, 4, 3, 0, 0, 0, 3, 4, 0, 0, 3, 4, 3, 2, 3, 4, 3, 0, 0, 0, 3, 4, 4, 4, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

7 0, 2, 2, 2, 2, 2, 2, 2, 0, 2, 4, 4, 4, 4, 4, 4, 4, 0, 0, 2, 2, 2, 2, 3, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 1, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 1, 4, 2, 0, 0, 0, 0, 0, 0, 0, 2, 4, 1, 0, 0, 0, 0, 0, 0, 0, 4, 3, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 1, 0, 0, 0, 0, 0, 0, 4, 3, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

8 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 4, 4, 2, 0, 0, 0, 0, 4, 4, 2, 3, 4, 2, 0, 0, 2, 4, 2, 0, 0, 4, 4, 0, 0, 2, 4, 2, 0, 0, 4, 4, 0, 0, 0, 4, 4, 2, 3, 4, 2, 0, 0, 0, 2, 4, 4, 4, 4, 0, 0, 0, 1, 4, 4, 2, 3, 4, 3, 0, 0, 3, 4, 1, 0, 0, 3, 4, 1, 0, 4, 4, 0, 0, 0, 2, 4, 2, 0, 2, 4, 1, 0, 0, 3, 4, 0, 0, 1, 4, 4, 2, 3, 4, 3, 0, 0, 0, 1, 4, 4, 4, 3, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0

9 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 4, 4, 4, 3, 0, 0, 0, 3, 4, 3, 2, 3, 4, 3, 0, 0, 4, 2, 0, 0, 0, 2, 4, 0, 0, 4, 2, 0, 0, 0, 2, 4, 1, 0, 4, 4, 0, 0, 0, 3, 4, 2, 0, 3, 4, 4, 4, 4, 4, 4, 2, 0, 0, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 3, 0, 0, 0, 0, 0, 0, 3, 4, 1, 0, 0, 0, 0, 0, 3, 4, 3, 0, 0, 0, 1, 2, 4, 4, 3, 0, 0, 0, 0, 1, 4, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

a 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 3, 4, 3, 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 1, 4, 4, 4, 1, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 4, 4, 0, 4, 4, 0, 0, 0, 0, 4, 4, 0, 4, 4, 0, 0, 0, 2, 4, 4, 4, 4, 4, 2, 0, 0, 3, 4, 2, 2, 2, 4, 3, 0, 0, 4, 3, 0, 0, 0, 3, 4, 0, 1, 2, 1, 0, 0, 0, 1, 2, 1

A 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 3, 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 0, 2, 4, 2, 4, 2, 0, 0, 0, 0, 0, 4, 4, 0, 4, 4, 0, 0, 0, 0, 0, 4, 2, 0, 2, 4, 0, 0, 0, 2, 4, 3, 2, 3, 4, 2, 0, 0, 2, 4, 4, 4, 4, 4, 2, 0, 0, 4, 4, 2, 2, 2, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 1, 2, 0

B 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 3, 1, 0, 0, 0, 4, 4, 2, 2, 4, 4, 1, 0, 0, 4, 2, 0, 0, 1, 4, 2, 0, 0, 4, 2, 0, 0, 0, 4, 2, 0, 0, 4, 3, 2, 2, 3, 4, 1, 0, 0, 4, 4, 4, 4, 4, 4, 0, 0, 0, 4, 3, 2, 2, 3, 4, 1, 0, 0, 4, 2, 0, 0, 0, 4, 3, 0, 0, 4, 2, 0, 0, 0, 2, 4, 0, 0, 4, 2, 0, 0, 1, 4, 3, 0, 0, 4, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 4, 4, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

C 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 3, 2, 2, 4, 4, 0, 1, 4, 3, 0, 0, 0, 2, 4, 1, 2, 4, 2, 0, 0, 0, 0, 2, 1, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 4, 3, 0, 0, 0, 1, 2, 0, 0, 4, 4, 3, 2, 2, 4, 4, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

D 1, 2, 2, 2, 0, 0, 0, 0, 0, 2, 4, 4, 4, 4, 3, 1, 0, 0, 2, 4, 3, 2, 3, 4, 4, 1, 0, 2, 4, 2, 0, 0, 1, 4, 4, 0, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 0, 4, 4, 2, 4, 2, 0, 0, 0, 0, 4, 4, 2, 4, 2, 0, 0, 0, 0, 4, 4, 2, 4, 2, 0, 0, 0, 0, 4, 4, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 1, 4, 4, 1, 2, 4, 3, 2, 3, 4, 4, 1, 0, 2, 4, 4, 4, 4, 3, 1, 0, 0, 1, 2, 2, 2, 0, 0, 0, 0, 0

E 0, 2, 2, 2, 2, 2, 2, 1, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 4, 4, 2, 2, 2, 2, 1, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 2, 2, 2, 2, 1, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 2, 2, 2, 2, 2, 2, 1, 0

F 0, 2, 2, 2, 2, 2, 2, 1, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 4, 4, 2, 2, 2, 2, 1, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0

G 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 4, 4, 3, 0, 0, 0, 1, 4, 3, 2, 3, 4, 2, 0, 0, 3, 4, 0, 0, 0, 3, 4, 0, 0, 4, 4, 0, 0, 0, 1, 2, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 4, 4, 4, 0, 0, 4, 4, 0, 0, 2, 4, 4, 0, 0, 4, 4, 0, 0, 0, 2, 4, 0, 0, 4, 4, 0, 0, 0, 2, 4, 0, 0, 2, 4, 3, 2, 2, 4, 3, 0, 0, 0, 3, 4, 4, 4, 3, 0, 0, 0, 0, 0, 1, 2, 2, 0, 0, 0

H 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0

I 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

J 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 1, 4, 4, 2, 2, 3, 4, 3, 0, 0, 1, 4, 4, 4, 4, 3, 0, 0, 0, 0, 1, 2, 2, 2, 0, 0, 0

K 0, 2, 2, 0, 0, 0, 1, 2, 1, 0, 4, 4, 0, 0, 0, 4, 4, 1, 0, 4, 4, 0, 0, 2, 4, 2, 0, 0, 4, 4, 0, 0, 4, 4, 1, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 4, 3, 0, 0, 0, 0, 4, 4, 4, 4, 2, 0, 0, 0, 0, 4, 4, 4, 4, 3, 0, 0, 0, 0, 4, 4, 0, 4, 4, 0, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 1, 4, 4, 0, 0, 0, 4, 4, 0, 0, 2, 4, 2, 0, 0, 4, 4, 0, 0, 1, 4, 3, 0, 0, 2, 2, 0, 0, 0, 2, 2, 1

L 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 2, 2, 2, 2, 2, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0

M 1, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 1, 2, 2, 4, 0, 0, 0, 0, 0, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 4, 1, 0, 1, 4, 4, 2, 2, 4, 4, 4, 0, 4, 4, 4, 2, 2, 4, 4, 4, 4, 4, 4, 4, 2, 2, 4, 3, 4, 4, 4, 3, 4, 2, 2, 4, 2, 2, 4, 2, 2, 4, 2, 2, 4, 2, 2, 4, 2, 0, 2, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 1, 2, 1, 0, 0, 0, 1, 2, 1

N 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 3, 0, 0, 0, 0, 4, 4, 0, 0, 4, 2, 0, 0, 0, 4, 4, 0, 0, 4, 4, 1, 0, 0, 4, 4, 0, 0, 4, 4, 4, 0, 0, 4, 4, 0, 0, 4, 4, 4, 3, 0, 4, 4, 0, 0, 4, 4, 4, 4, 1, 4, 4, 0, 0, 4, 4, 1, 4, 4, 4, 4, 0, 0, 4, 4, 0, 3, 4, 4, 4, 0, 0, 4, 4, 0, 0, 4, 4, 4, 0, 0, 4, 4, 0, 0, 1, 4, 4, 0, 0, 4, 4, 0, 0, 0, 2, 4, 0, 0, 4, 4, 0, 0, 0, 0, 3, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0

o 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 1, 3, 4, 4, 4, 3, 1, 0, 0, 3, 4, 2, 2, 2, 4, 3, 0, 1, 4, 2, 0, 0, 0, 2, 4, 1, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 1, 4, 2, 0, 0, 0, 2, 4, 1, 0, 3, 4, 2, 2, 2, 4, 3, 0, 0, 1, 3, 4, 4, 4, 3, 1, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

O 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 3, 2, 3, 4, 4, 0, 1, 4, 3, 0, 0, 0, 3, 4, 1, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 1, 4, 3, 0, 0, 0, 3, 4, 1, 0, 4, 4, 3, 2, 3, 4, 4, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

P 0, 2, 2, 2, 2, 1, 0, 0, 0, 0, 4, 4, 4, 4, 4, 3, 0, 0, 0, 4, 4, 3, 2, 3, 4, 3, 0, 0, 4, 4, 0, 0, 0, 3, 4, 0, 0, 4, 4, 0, 0, 0, 2, 4, 0, 0, 4, 4, 2, 2, 2, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 2, 2, 2, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0

Q 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 3, 2, 3, 4, 4, 0, 1, 4, 3, 0, 0, 0, 3, 4, 1, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 1, 0, 2, 4, 2, 2, 4, 2, 0, 2, 3, 3, 4, 2, 1, 4, 3, 0, 2, 4, 4, 4, 2, 0, 4, 4, 3, 2, 4, 4, 4, 4, 0, 1, 4, 4, 4, 4, 4, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0

R 0, 2, 2, 2, 2, 1, 0, 0, 0, 0, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 3, 2, 3, 4, 3, 0, 0, 4, 4, 0, 0, 0, 3, 4, 0, 0, 4, 4, 0, 0, 0, 2, 4, 0, 0, 4, 4, 2, 0, 2, 4, 3, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 4, 4, 2, 3, 4, 3, 0, 0, 0, 4, 4, 0, 1, 4, 3, 0, 0, 0, 4, 4, 0, 0, 2, 4, 0, 0, 0, 4, 4, 0, 0, 2, 4, 2, 0, 0, 4, 4, 0, 0, 1, 4, 4, 0, 0, 4, 4, 0, 0, 0, 3, 4, 2, 0, 2, 2, 0, 0, 0, 1, 2, 1

S 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 2, 2, 2, 3, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 0, 4, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 4, 4, 4, 2, 1, 0, 0, 0, 1, 2, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 1, 4, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 1, 0, 3, 2, 0, 0, 2, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 1, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0

T 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 3, 4, 3, 2, 2, 1, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

U 1, 2, 1, 0, 0, 0, 2, 2, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 0, 4, 3, 2, 2, 2, 4, 2, 0, 0, 1, 4, 4, 4, 4, 3, 0, 0, 0, 0, 1, 2, 2, 2, 0, 0, 0

V 1, 2, 1, 0, 0, 0, 1, 2, 1, 0, 4, 2, 0, 0, 0, 2, 4, 1, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 2, 4, 1, 0, 1, 4, 3, 0, 0, 2, 4, 2, 0, 2, 4, 2, 0, 0, 1, 4, 2, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 4, 4, 0, 0, 0, 0, 3, 4, 0, 4, 4, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 1, 4, 4, 4, 2, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

W 1, 2, 1, 0, 0, 0, 1, 2, 1, 2, 4, 2, 0, 0, 0, 2, 4, 2, 0, 4, 2, 0, 0, 0, 2, 4, 0, 0, 4, 2, 0, 0, 0, 2, 4, 0, 0, 4, 2, 0, 0, 0, 2, 4, 0, 0, 4, 3, 0, 0, 0, 3, 4, 0, 0, 3, 4, 0, 2, 0, 4, 3, 0, 0, 2, 4, 1, 4, 1, 4, 2, 0, 0, 2, 4, 4, 4, 4, 4, 2, 0, 0, 2, 4, 4, 4, 4, 4, 2, 0, 0, 2, 4, 3, 0, 3, 4, 2, 0, 0, 0, 4, 2, 0, 2, 4, 0, 0, 0, 0, 4, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

X 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 3, 4, 2, 0, 2, 4, 3, 0, 0, 1, 4, 3, 0, 3, 4, 1, 0, 0, 0, 3, 4, 2, 4, 3, 0, 0, 0, 0, 1, 4, 4, 4, 1, 0, 0, 0, 0, 0, 3, 4, 3, 0, 0, 0, 0, 0, 0, 3, 4, 3, 0, 0, 0, 0, 0, 1, 4, 4, 4, 1, 0, 0, 0, 0, 3, 4, 2, 4, 3, 0, 0, 0, 0, 4, 4, 0, 3, 4, 1, 0, 0, 2, 4, 2, 0, 2, 4, 3, 0, 0, 4, 4, 0, 0, 0, 3, 4, 0, 0, 2, 1, 0, 0, 0, 1, 2, 0

Y 1, 2, 1, 0, 0, 0, 1, 2, 1, 0, 4, 4, 1, 0, 1, 4, 4, 0, 0, 2, 4, 2, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 4, 4, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

Z 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 2, 2, 2, 2, 3, 4, 4, 0, 0, 0, 0, 0, 0, 3, 4, 2, 0, 0, 0, 0, 0, 1, 4, 3, 0, 0, 0, 0, 0, 0, 3, 4, 1, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 1, 4, 3, 0, 0, 0, 0, 0, 0, 3, 4, 1, 0, 0, 0, 0, 0, 2, 4, 3, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 2, 2, 2, 2, 2, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0

0 0, 0, 1, 2, 2, 2, 1, 0, 0, 0, 3, 4, 4, 4, 4, 4, 3, 0, 1, 4, 4, 4, 4, 4, 4, 4, 1, 2, 4, 4, 1, 0, 1, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 1, 0, 1, 4, 4, 2, 1, 4, 4, 4, 4, 4, 4, 4, 1, 0, 3, 4, 4, 4, 4, 4, 3, 0, 0, 0, 1, 2, 2, 2, 1, 0, 0

1 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

2 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 3, 4, 4, 4, 4, 1, 0, 0, 3, 4, 4, 4, 4, 4, 4, 4, 0, 2, 4, 4, 2, 0, 1, 4, 4, 0, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 2, 0, 0, 3, 4, 4, 0, 0, 2, 0, 0, 3, 4, 4, 3, 0, 0, 0, 0, 3, 4, 4, 3, 0, 0, 0, 0, 3, 4, 4, 3, 0, 0, 0, 0, 0, 1, 4, 4, 3, 0, 0, 0, 0, 0, 3, 4, 4, 0, 0, 0, 0, 0, 1, 4, 4, 3, 2, 2, 2, 2, 2, 0, 0, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 1, 2, 2, 2, 2, 2, 2, 2, 0

3 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 2, 2, 2, 3, 4, 4, 2, 0, 0, 0, 0, 0, 3, 4, 4, 0, 0, 0, 0, 0, 2, 4, 4, 2, 0, 0, 0, 0, 1, 4, 4, 4, 3, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 3, 4, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 4, 1, 0, 1, 4, 4, 2, 0, 4, 4, 4, 4, 4, 4, 4, 1, 0, 1, 4, 4, 4, 4, 4, 4, 1, 0, 0, 0, 0, 2, 2, 2, 1, 0, 0

4 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 4, 4, 2, 0, 0, 0, 0, 0, 1, 4, 4, 1, 0, 0, 0, 0, 0, 2, 4, 3, 0, 0, 0, 0, 0, 0, 4, 4, 2, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 3, 4, 2, 2, 4, 4, 0, 0, 2, 4, 4, 1, 2, 4, 4, 0, 0, 3, 4, 4, 0, 2, 4, 4, 1, 2, 4, 4, 4, 4, 4, 4, 4, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 1, 2, 2, 0

5 0, 0, 1, 2, 2, 2, 2, 2, 0, 0, 0, 3, 4, 4, 4, 4, 4, 0, 0, 0, 4, 4, 4, 4, 4, 4, 3, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 2, 1, 0, 0, 0, 0, 4, 4, 4, 4, 4, 3, 1, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 4, 1, 0, 1, 4, 4, 2, 1, 4, 4, 4, 4, 4, 4, 4, 1, 0, 1, 4, 4, 4, 4, 4, 4, 1, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0

6 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 1, 3, 4, 4, 2, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 1, 4, 4, 4, 4, 2, 0, 0, 0, 0, 3, 4, 3, 0, 0, 0, 0, 0, 1, 4, 4, 3, 2, 2, 0, 0, 0, 2, 4, 4, 4, 4, 4, 4, 1, 0, 2, 4, 4, 4, 4, 4, 4, 4, 0, 2, 4, 4, 1, 0, 1, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 1, 4, 4, 3, 2, 3, 4, 4, 0, 0, 1, 4, 4, 4, 4, 4, 4, 1, 0, 0, 0, 1, 2, 2, 2, 0, 0, 0

7 1, 2, 2, 2, 2, 2, 2, 2, 0, 2, 4, 4, 4, 4, 4, 4, 4, 0, 1, 2, 2, 2, 2, 3, 4, 4, 0, 0, 0, 0, 0, 0, 3, 4, 2, 0, 0, 0, 0, 0, 0, 4, 4, 1, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 2, 0, 0, 0, 0, 0, 1, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 2, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 3, 4, 3, 0, 0, 0, 0, 0, 1, 4, 4, 1, 0, 0, 0, 0, 0, 1, 2, 2, 0, 0, 0, 0, 0

8 0, 0, 0, 1, 2, 2, 0, 0, 0, 0, 0, 3, 4, 4, 4, 4, 1, 0, 0, 1, 4, 4, 4, 4, 4, 3, 0, 0, 2, 4, 3, 0, 1, 4, 4, 0, 0, 2, 4, 3, 0, 1, 4, 4, 0, 0, 1, 4, 4, 4, 4, 4, 3, 0, 0, 0, 4, 4, 4, 4, 4, 2, 0, 0, 2, 4, 4, 4, 4, 4, 4, 4, 0, 0, 4, 4, 3, 0, 1, 4, 4, 2, 0, 4, 4, 2, 0, 0, 4, 4, 2, 0, 4, 4, 3, 0, 1, 4, 4, 2, 0, 2, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 3, 4, 4, 4, 4, 1, 0, 0, 0, 0, 2, 2, 2, 1, 0, 0

9 0, 0, 0, 2, 2, 2, 1, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 3, 2, 3, 4, 4, 1, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 3, 4, 2, 2, 4, 4, 1, 0, 1, 4, 4, 2, 0, 4, 4, 4, 4, 4, 4, 4, 2, 0, 1, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 2, 2, 3, 4, 4, 1, 0, 0, 0, 0, 0, 3, 4, 3, 0, 0, 0, 0, 2, 4, 4, 4, 1, 0, 0, 1, 4, 4, 4, 4, 1, 0, 0, 0, 2, 4, 4, 3, 1, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 0

a 0, 2, 2, 1, 0, 1, 2, 2, 0, 0, 4, 4, 2, 0, 2, 4, 4, 0, 0, 4, 4, 2, 0, 2, 4, 4, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 1, 4, 4, 4, 1, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 2, 4, 4, 0, 4, 4, 2, 0, 0, 3, 4, 4, 4, 4, 4, 3, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 2, 4, 4, 0, 0, 0, 4, 4, 2, 1, 2, 2, 0, 0, 0, 2, 2, 1

A 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 3, 4, 4, 4, 3, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 0, 0, 0, 1, 4, 4, 0, 4, 4, 1, 0, 0, 2, 4, 4, 0, 4, 4, 2, 0, 0, 3, 4, 4, 4, 4, 4, 3, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 0, 1, 4, 4, 4, 4, 4, 4, 4, 1, 2, 4, 4, 1, 0, 1, 4, 4, 2, 1, 2, 2, 0, 0, 0, 2, 2, 1

B 1, 2, 2, 2, 2, 1, 0, 0, 0, 2, 4, 4, 4, 4, 4, 3, 0, 0, 2, 4, 4, 4, 4, 4, 4, 4, 3, 0, 2, 4, 4, 4, 0, 0, 3, 4, 4, 0, 2, 4, 4, 0, 0, 2, 4, 4, 0, 2, 4, 4, 2, 2, 4, 4, 3, 0, 2, 4, 4, 4, 4, 4, 4, 2, 0, 2, 4, 4, 4, 4, 4, 4, 3, 0, 2, 4, 4, 0, 0, 3, 4, 4, 1, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 3, 4, 4, 1, 2, 4, 4, 4, 4, 4, 4, 4, 3, 0, 2, 4, 4, 4, 4, 4, 4, 3, 0, 0, 1, 2, 2, 2, 2, 1, 0, 0, 0

C 0, 0, 1, 2, 2, 2, 1, 0, 0, 0, 3, 4, 4, 4, 4, 4, 1, 0, 1, 4, 4, 4, 4, 4, 4, 4, 1, 2, 4, 4, 1, 0, 0, 3, 4, 2, 2, 4, 4, 0, 0, 0, 1, 4, 2, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 1, 0, 0, 3, 4, 1, 1, 4, 4, 4, 4, 4, 4, 4, 2, 0, 3, 4, 4, 4, 4, 4, 3, 0, 0, 0, 1, 2, 2, 2, 2, 0, 0

D 1, 2, 2, 2, 2, 1, 0, 0, 0, 2, 4, 4, 4, 4, 4, 3, 0, 0, 2, 4, 4, 4, 4, 4, 4, 4, 3, 0, 2, 4, 4, 0, 1, 3, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 4, 2, 4, 4, 0, 0, 0, 2, 4, 4, 2, 4, 4, 0, 0, 0, 2, 4, 4, 2, 4, 4, 0, 0, 0, 2, 4, 4, 2, 4, 4, 0, 0, 0, 4, 4, 4, 2, 4, 4, 0, 1, 3, 4, 4, 2, 2, 4, 4, 4, 4, 4, 4, 4, 3, 0, 2, 4, 4, 4, 4, 4, 3, 0, 0, 1, 2, 2, 2, 2, 1, 0, 0, 0

E 0, 2, 2, 2, 2, 2, 2, 1, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 4, 4, 2, 2, 2, 2, 1, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 2, 0, 0, 2, 2, 2, 2, 2, 2, 1, 0

F 0, 2, 2, 2, 2, 2, 2, 1, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 4, 4, 2, 2, 2, 2, 1, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0

G 0, 0, 0, 2, 2, 2, 1, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 3, 4, 4, 4, 4, 4, 3, 0, 0, 4, 4, 3, 0, 1, 4, 4, 2, 0, 4, 4, 0, 0, 0, 3, 4, 1, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 2, 2, 2, 0, 0, 4, 4, 0, 0, 4, 4, 4, 2, 0, 4, 4, 0, 0, 4, 4, 4, 2, 0, 4, 4, 0, 0, 0, 4, 4, 2, 0, 4, 4, 1, 0, 0, 4, 4, 2, 0, 4, 4, 4, 4, 4, 4, 4, 1, 0, 1, 4, 4, 4, 4, 4, 3, 0, 0, 0, 1, 2, 2, 2, 2, 0, 0

H 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 2, 2, 2, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 4, 4, 2, 2, 2, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 2, 2, 0, 0, 0, 2, 2, 0

I 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

J 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 2, 4, 4, 0, 0, 1, 4, 4, 0, 2, 4, 4, 3, 2, 4, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 2, 2, 2, 2, 1, 0, 0

K 1, 2, 2, 0, 0, 0, 1, 2, 1, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 2, 4, 3, 0, 2, 4, 4, 0, 1, 4, 4, 1, 0, 2, 4, 4, 0, 3, 4, 3, 0, 0, 2, 4, 4, 2, 4, 4, 0, 0, 0, 2, 4, 4, 4, 4, 4, 0, 0, 0, 2, 4, 4, 4, 4, 4, 0, 0, 0, 2, 4, 4, 2, 4, 4, 2, 0, 0, 2, 4, 4, 0, 3, 4, 4, 0, 0, 2, 4, 4, 0, 2, 4, 4, 2, 0, 2, 4, 4, 0, 0, 4, 4, 4, 0, 2, 4, 4, 0, 0, 2, 4, 4, 2, 1, 2, 2, 0, 0, 0, 2, 2, 1

L 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 2, 2, 2, 2, 2, 0, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0

M 1, 1, 0, 0, 0, 0, 0, 1, 1, 2, 3, 0, 0, 0, 0, 0, 3, 2, 2, 4, 1, 0, 0, 0, 1, 4, 2, 2, 4, 3, 0, 0, 0, 3, 4, 2, 2, 4, 4, 2, 0, 2, 4, 4, 2, 2, 4, 4, 4, 4, 4, 4, 4, 2, 2, 4, 4, 4, 4, 4, 4, 4, 2, 2, 4, 4, 4, 4, 4, 4, 4, 2, 2, 4, 3, 3, 4, 3, 3, 4, 2, 2, 4, 2, 0, 2, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 1, 2, 1, 0, 0, 0, 1, 2, 1

N 0, 2, 0, 0, 0, 0, 2, 2, 0, 0, 4, 1, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 3, 0, 0, 4, 4, 0, 0, 4, 4, 4, 1, 0, 4, 4, 0, 0, 4, 4, 4, 4, 0, 4, 4, 0, 0, 4, 4, 4, 4, 3, 4, 4, 0, 0, 4, 4, 3, 4, 4, 4, 4, 0, 0, 4, 4, 0, 4, 4, 4, 4, 0, 0, 4, 4, 0, 1, 4, 4, 4, 0, 0, 4, 4, 0, 0, 3, 4, 4, 0, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 0, 0, 1, 4, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0

o 1, 2, 2, 0, 0, 0, 2, 2, 1, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 0, 0, 1, 2, 2, 2, 1, 0, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 4, 2, 4, 4, 4, 0, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 1, 4, 4, 4, 2, 4, 4, 4, 1, 0, 2, 4, 4, 4, 4, 4, 2, 0, 0, 0, 1, 2, 2, 2, 1, 0, 0

O 0, 0, 1, 2, 2, 2, 1, 0, 0, 0, 3, 4, 4, 4, 4, 4, 3, 0, 1, 4, 4, 4, 4, 4, 4, 4, 1, 2, 4, 4, 1, 0, 1, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 1, 0, 1, 4, 4, 2, 1, 4, 4, 4, 4, 4, 4, 4, 1, 0, 3, 4, 4, 4, 4, 4, 3, 0, 0, 0, 1, 2, 2, 2, 1, 0, 0

P 0, 2, 2, 2, 2, 2, 1, 0, 0, 0, 4, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 4, 4, 4, 4, 4, 1, 0, 4, 4, 2, 0, 1, 4, 4, 2, 0, 4, 4, 2, 0, 0, 4, 4, 2, 0, 4, 4, 3, 2, 3, 4, 4, 2, 0, 4, 4, 4, 4, 4, 4, 3, 0, 0, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0

Q 0, 0, 1, 2, 2, 2, 1, 0, 0, 0, 3, 4, 4, 4, 4, 4, 3, 0, 1, 4, 4, 4, 4, 4, 4, 4, 1, 2, 4, 4, 1, 0, 1, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 2, 4, 4, 0, 1, 1, 4, 4, 2, 2, 4, 4, 0, 2, 4, 4, 4, 2, 2, 4, 4, 1, 2, 4, 4, 4, 3, 1, 4, 4, 4, 4, 4, 4, 4, 4, 0, 3, 4, 4, 4, 4, 4, 3, 4, 0, 0, 1, 2, 2, 2, 1, 0, 1

R 0, 2, 2, 2, 2, 2, 1, 0, 0, 0, 4, 4, 4, 4, 4, 4, 1, 0, 0, 4, 4, 4, 4, 4, 4, 4, 1, 0, 4, 4, 2, 0, 1, 4, 4, 2, 0, 4, 4, 2, 0, 0, 4, 4, 2, 0, 4, 4, 3, 2, 3, 4, 4, 2, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 0, 0, 0, 4, 4, 0, 3, 4, 4, 0, 0, 0, 4, 4, 0, 1, 4, 4, 1, 0, 0, 4, 4, 0, 0, 3, 4, 3, 0, 0, 4, 4, 0, 0, 2, 4, 4, 1, 0, 4, 4, 0, 0, 0, 4, 4, 3, 0, 2, 2, 0, 0, 0, 1, 2, 2

S 0, 0, 2, 2, 2, 2, 1, 0, 0, 0, 3, 4, 4, 4, 4, 4, 2, 0, 1, 4, 4, 2, 2, 4, 4, 1, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 4, 3, 2, 0, 0, 0, 0, 0, 3, 4, 4, 4, 4, 4, 1, 0, 0, 0, 2, 4, 4, 4, 4, 4, 1, 0, 0, 0, 0, 0, 2, 4, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 3, 4, 2, 0, 4, 4, 2, 2, 3, 4, 4, 2, 0, 4, 4, 4, 4, 4, 4, 3, 0, 0, 1, 2, 2, 2, 2, 2, 0, 0

T 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 3, 4, 3, 2, 2, 1, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

U 1, 2, 1, 0, 0, 0, 2, 2, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 2, 0, 0, 0, 4, 4, 0, 2, 4, 3, 0, 0, 1, 4, 4, 0, 2, 4, 4, 3, 2, 4, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 2, 2, 2, 2, 1, 0, 0

V 1, 2, 1, 0, 0, 0, 1, 2, 1, 2, 4, 3, 0, 0, 0, 3, 4, 2, 1, 4, 4, 0, 0, 0, 4, 4, 2, 0, 4, 4, 2, 0, 2, 4, 4, 0, 0, 3, 4, 3, 0, 3, 4, 4, 0, 0, 2, 4, 4, 0, 4, 4, 2, 0, 0, 2, 4, 4, 0, 4, 4, 2, 0, 0, 0, 4, 4, 2, 4, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 0, 4, 4, 4, 1, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

W 1, 2, 1, 0, 0, 0, 1, 2, 1, 2, 4, 2, 0, 0, 0, 2, 4, 2, 2, 4, 3, 0, 0, 0, 3, 4, 2, 2, 4, 4, 0, 0, 0, 4, 4, 2, 1, 4, 4, 0, 0, 0, 4, 4, 1, 0, 4, 4, 0, 0, 0, 4, 4, 0, 0, 4, 4, 0, 4, 4, 0, 4, 0, 4, 4, 4, 0, 0, 4, 4, 3, 4, 3, 4, 4, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 2, 4, 4, 4, 4, 4, 2, 0, 0, 2, 4, 4, 2, 4, 4, 2, 0, 0, 2, 4, 3, 0, 3, 4, 2, 0, 0, 2, 4, 2, 0, 2, 4, 2, 0, 0, 0, 2, 1, 0, 1, 2, 0, 0

X 1, 2, 2, 0, 0, 0, 2, 2, 1, 2, 4, 4, 1, 0, 1, 4, 4, 2, 0, 4, 4, 3, 0, 3, 4, 4, 0, 0, 2, 4, 4, 0, 4, 4, 2, 0, 0, 0, 4, 4, 4, 4, 4, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 0, 0, 4, 4, 4, 1, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 2, 4, 4, 4, 2, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 0, 0, 0, 2, 4, 4, 0, 4, 4, 2, 0, 0, 4, 4, 3, 0, 3, 4, 4, 0, 2, 4, 4, 1, 0, 1, 4, 4, 2, 1, 2, 2, 0, 0, 0, 2, 2, 1

Y 1, 2, 2, 0, 0, 0, 2, 2, 1, 1, 4, 4, 2, 0, 2, 4, 4, 1, 0, 3, 4, 4, 0, 4, 4, 3, 0, 0, 1, 4, 4, 4, 4, 4, 1, 0, 0, 0, 3, 4, 4, 4, 3, 0, 0, 0, 0, 1, 4, 4, 4, 2, 0, 0, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 2, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0

Z 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 2, 2, 4, 4, 4, 2, 0, 0, 0, 0, 1, 4, 4, 3, 0, 0, 0, 0, 0, 3, 4, 4, 1, 0, 0, 0, 0, 1, 4, 4, 3, 0, 0, 0, 0, 0, 0, 3, 4, 4, 1, 0, 0, 0, 0, 3, 4, 4, 1, 0, 0, 0, 0, 0, 1, 4, 4, 2, 0, 0, 0, 0, 0, 4, 4, 3, 1, 0, 0, 0, 0, 0, 2, 4, 4, 2, 0, 0, 0, 0, 0, 0, 2, 4, 4, 3, 2, 2, 2, 2, 1, 2, 4, 4, 4, 4, 4, 4, 4, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1

A.3 The Accompanying CD

There are four directories in the accompanying CD as each of the modules has its own.

- ImageProcLib contains the image processing library.
- NeuralNetwork contains the neural network library.
- ImageSaver contains a library capable of saving JPEG-images.
- LicRecognizer is the library for application's user interface.

Each of these folders contain src, inc and sis directories. Src contains the source code, inc contains the header files and sis contains the installable SIS files. By importing these files into an integrated development environment of choice, one can compile and run the software on desired hardware. Instructions for installing the needed SDK and IDE can be found at website http://forum.nokia.com/main/resources/tools_and_sdks/index.html#cpp.

In the root of the CD there is a PDF-file containing the thesis.